NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EVALUATING THE LIMITS OF NETWORK TOPOLOGY INFERENCE VIA VIRTUALIZED NETWORK EMULATION

by

Erik C. Rye

June 2015

Thesis Co-Advisors:	Robert Beverly
	Ralucca Gera
Second Reader:	Justin Rohrer

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE 06-19-2015	3. REPORT TYPE AND DATES COVERED Master's Thesis 07-08-2013 to 06-19-2015

4. TITLE AND SUBTITLE

EVALUATING THE LIMITS OF NETWORK TOPOLOGY INFERENCE VIA VIRTUALIZED NETWORK EMULATION

5. FUNDING NUMBERS

N66001-2250-58231

6. AUTHOR(S)

Erik C. Rye

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Homeland Security
245 Murray Lane SW, Washington, DC 20528

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The Internet measurement community is beset by a lack of "ground truth," or knowledge of the real, underlying network in topology inference experiments. While better tools and methodologies can be developed, quantifying the effectiveness of these mapping utilities and explaining pathologies is difficult, if not impossible, without knowing the network topology being probed. In this thesis we present a tool that eliminates topological uncertainty in an emulated, virtualized environment. First, we automatically build topological ground truth according to various network generation models and create emulated Cisco router networks by leveraging and modifying existing emulation software. We then automate topological inference from one vantage point at a time for every vantage point in the network. Finally, we incorporate a mechanism to study common sources of network topology inference abnormalities by including the ability to induce link failures within the network. In addition, this thesis reexamines previous work in sampling Autonomous System-level Internet graphs to procure realistic models for emulation and simulation. We build upon this work by including additional data sets, and more recent Internet topologies to sample from, and observe divergent results from the authors of the original work. Lastly, we introduce a new technique for sampling Internet graphs that better retains particular graph metrics across multiple timeframes and data sets.

14. SUBJECT TERMS

Network Emulation, Graph Sampling, Topology Inference, Network Measurement

15. NUMBER OF PAGES 115

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EVALUATING THE LIMITS OF NETWORK TOPOLOGY INFERENCE VIA VIRTUALIZED NETWORK EMULATION

Erik C. Rye
Captain, United States Marine Corps
B.S., U.S. Naval Academy, 2008

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN COMPUTER SCIENCE

and

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2015

Author:	Erik C. Rye
Approved by:	Robert Beverly Thesis Co-Advisor
	Ralucca Gera Thesis Co-Advisor
	Justin Rohrer Second Reader
	Peter Denning Chair, Department of Computer Science
	Craig Rasmussen Chair, Department of Applied Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Internet measurement community is beset by a lack of “ground truth,” or knowledge of the real, underlying network in topology inference experiments. While better tools and methodologies can be developed, quantifying the effectiveness of these mapping utilities and explaining pathologies is difficult, if not impossible, without knowing the network topology being probed. In this thesis we present a tool that eliminates topological uncertainty in an emulated, virtualized environment. First, we automatically build topological ground truth according to various network generation models and create emulated Cisco router networks by leveraging and modifying existing emulation software. We then automate topological inference from one vantage point at a time for every vantage point in the network. Finally, we incorporate a mechanism to study common sources of network topology inference abnormalities by including the ability to induce link failures within the network. In addition, this thesis reexamines previous work in sampling Autonomous System-level Internet graphs to procure realistic models for emulation and simulation. We build upon this work by including additional data sets, and more recent Internet topologies to sample from, and observe divergent results from the authors of the original work. Lastly, we introduce a new technique for sampling Internet graphs that better retains particular graph metrics across multiple timeframes and data sets.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

Ark	Archipelago
AS	Autonomous System
ASN	Autonomous System Number
BA	Barabási-Albert
BGP	Border Gateway Protocol
CAIDA	Center for Applied Internet Data Analysis
CDF	Cumulative Distribution Function
CDN	Content Distribution Network
CPU	Central Processing Unit
CRE	Contraction of a Random Edge
CRVE	Contraction of a Random Vertex-Edge
DARPA	Defense Advanced Research Projects Agency
DNS	Domain Name System
DRE	Deletion of a Random Edge
DRV	Deletion of a Random Vertex
DRVE	Deletion of a Random Vertex-Edge
DHYB	Deletion Hybrid
EBFS	Exploration by Breadth-First Search
EDFS	Exploration by Depth-First Search
ERIK	Emulated Router Inference Kit

FIB	Forwarding Information Base
GNS3	Graphical Network Simulator - 3
ICMP	Internet Control Message Protocol
IOS	Internetwork Operating System
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISP	Internet Service Provider
IXP	Internet Exchange Point
KDD	*k*-core decomposition/DRVE/DRE
KKD	*k*-core decomposition/*k*-deletion/DRE
MAE	mean absolute error
MERLIN	MEasure the Router Level of the INternet
MIPS	Microprocessor without Interlocked Pipeline Stages
OS	operating system
OSPF	Open Shortest-Path First
PoP	Point-of-Presence
RIB	Routing Information Base
RIP	Routing Information Protocol
RIPE	Réseaux IP Européens
SYN	synchronization

TCP	Transmission Control Protocol
TTL	Time-to-Live
UDP	User Datagram Protocol
UI	user interface
UML	User-Mode Linux
VIRL	Virtual Internet Routing Lab
VM	virtual machine

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

First and foremost, I would like to acknowledge my wife Andrea's support throughout our time here at the Naval Postgraduate School. I truly could not have done it without her, and I am deeply grateful for all of the sacrifices she made over the past two years. I am also forever thankful my parents encouraged my love for science and learning.

My advisors, Rob Beverly and Ralucca Gera, as well as my reader Justin Rohrer, all provided me with an immeasurable amount of guidance, patience, and wisdom throughout my time as a student. Rob, in addition to giving me an enormous amount of trust and latitude to do real science, often provided sound "life advising," and somehow got me through my first marathon. I will always appreciate Ralucca for believing in me, and for broadening my mathematical horizon into graph theory and network science — a world I didn't even know existed. Justin's firm grasp of all things Internet and knack for asking hard questions undoubtedly shaped the work I did here into a better product, although he did cause me to form an irrational grudge against `Matplotlib`.

Many professors who were not directly involved in my thesis went above and beyond in encouraging me throughout my studies at NPS, and I would be remiss without mentioning them here. Peter Denning, Loren Peitso, Geoff Xie, and Michael McCarrin of the Computer Science Department, and Pante Stanica, Carlos Borges, Craig Rasmussen, and David Canright from the Applied Math Department, all influenced me in a positive way, making me both a better student and individual. Many thanks to Jon Roginski, Thor Martinsen, and Brian Kropa for providing solid examples of what I'd like to do next.

My classmates were a phenomenal source of knowledge and assistance throughout our time together; I am appreciative of the friendships I made here that will doubtlessly continue long after we have departed Monterey.

Finally, I would like to note that much of this work could not have been done without the assistance of the *Dynamips* developers, who worked with us in quickly addressing issues we experienced. I appreciate their dedication and the quality software they maintain.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: Introduction

Internet measurement is an important facet of the ubiquitous resource that has transformed the way society conducts business, communicates, and consumes entertainment. Ascertaining the way that the Internet is connected and organized, however, is a difficult undertaking.

In seeking to ascertain the design and structure of the interconnected devices, enterprises, and organizations that constitute the Internet, the measurement community aims to expose its underlying topology. With a better understanding of the topology of the Internet, design choices can be made to increase the reliability of the network, its resiliency to attack or disruption, and improve the throughput and latency of traffic. Weaknesses can be identified and addressed, resulting a more robust network with an enhanced user experience. Further, longitudinal studies are important to our understanding of the drivers of Internet growth and our ability to make predictions about future Internet development. Network measurement is essential to activities from identifying Internet censorship under oppressive regimes and tracking the Internet's penetration into previously unserved countries and markets, to verifying compliance with policy, as in recent "network neutrality" rules established in the United States.

The Internet is a network of networks; Internet Service Providers (ISPs), and more generally, Autonomous Systems (ASs), are privately or government-controlled entities that form the basic building blocks of the global Internet as a whole, each with their own competing interests. In contrast to managed networks, like roadways and rail lines, there is no governing authority for the Internet that manages the complex web of fiscal, political, and engineering considerations that motivate the establishment of connections between ASs. Thus, there is no central body to appeal for a complete picture of the Internet's structure; while ASs are aware of the structure of their own networks, they are generally unwilling to divulge their network topology, because it is their source of revenue and could leave them vulnerable to attack by nefarious actors. Operators themselves may not know the entirety of their own topology, due to the magnitude of interconnections and rapidity with which large networks change.

The Internet was never designed to be measured; when conceived by the precursor to today's Defense Advanced Research Projects Agency (DARPA), its designers put a premium on resiliency under hardware and link failures [1]. Lacking a prescribed manner by which to directly measure Internet topology, the community largely relies on inferences using tools that imperfectly reveal topological information. Further complicating Internet measurement is the sheer number of physical devices that comprise it, and the dynamics induced by this hardware and the routing algorithms they employ. At the time of writing, the Internet is comprised of more than 46,000 ASs, a more than 1300% increase from the approximately 3200 that were present in 1997 when Border Gateway Protocol (BGP) Routing Information Base (RIB) data began to be compiled by the RouteViews project hosted by the University of Oregon [2].

The exponential growth and rapidly changing structure of the Internet are due to a wide variety of factors. The Internet is at its most fundamental level a collection of hardware devices and the physical media that interconnect them; electricity may be shut off, hardware components fail, and the cables between devices can be severed. These events are rarely predictable but frequently occur and generate changes in paths available for Internet data to traverse, which in turn affect measurement studies. Further, the paths that Internet routing devices choose for received data are a function of routing protocols and policies, which are chosen and implemented by network operators. Geographical proximity plays a role in determining device interconnection as well; it is often more convenient and cost-effective to connect routers that are closer together than to establish physical media links between physically disparate devices. Preexisting infrastructure often plays a role in the formation of connections between ASs; high-speed fiber optic cables are run along interstates and railroad tracks to connect major cities and are bundled together to cross the Atlantic. This leads to the formation of Points-of-Presence (PoP), Internet Exchange Points (IXPs), and carrier hosting facilities where many Internet connections converge.

Measurement experiments are subject to the effects of these design choices. At a higher level, the linkages between ASs are primarily motivated by economic concerns and profit. For example, the operator of an AS may enter into a customer-provider relationship with another AS because of an advantageous cost to transit their traffic, or form a "settlement-free" peering link with another comparably sized AS to avoid paying their provider for traffic between the two entities. These business relationships affect the paths available for Internet traffic to traverse and often result in traffic generated from different sources on the Internet being routed to the same destinations differently.

1.1 Motivation

The primary motivation in this thesis is to remove the limitations placed on network measurement experiments by the lack of an understanding of the "ground truth", or underlying topology of the network being measured. Whether inferring a subnetwork or the entire Internet, the measurement community relies on topology inference tools that will be discussed in more detail in Chapter 2. While these inference tools have been developed and adapted for this particular purpose, they also suffer from many limitations. More fundamentally, the accuracy of topology inference tools is difficult to evaluate, as a comparison against the true topology can only be made if the network structure is known, and the true topology is rarely known. While experiments against networks owned by the experimenter, or those with publicly published topologies, e.g., Internet2, can be performed, these networks are often small and immune to the complex policy and inter-AS interactions seen on the broader Internet. Toward that end, we endeavored to create our own ground truth by creating virtualized networks in an emulated environment, then using the state-of-the-art inference tools to discern the differences between inferred and true topologies.

Further, because we are creating our own network structure in an emulated environment, we endeavored to do this in a way that most realistically reflects true Internet characteristics. We therefore study previous experiments performed with this goal in mind, but broaden their results with more current data and from sources previously not considered.

1.2 Research Questions and Contributions

In our work, we investigate these primary questions:

1. Can we build an automated tool for network topology inference that combines network creation, emulation, and inference testing?
2. With our automated tool, do we discover interesting topological anomalies during inference experimentation?

3. Can we extend prior work that investigates Internet graph reduction techniques to other and more recent sources of Internet topology data? Do we obtain the same results?
4. Can we leverage graph characteristics to develop a novel reduction algorithm that performs well across data sets, if it is discovered that prior work algorithms do not?

We believe that we contribute to and extend the state-of-the-art through the following contributions:

1. We introduce a novel software tool that generates topological ground truth, emulates this network, and infers the topology using state-of-the-art network inference utility. Our program allows for several user defined parameters, including graph generation algorithm, number of routers, and link failure scenarios.
2. Our topology inference tool discovers anomalous routing behavior; we describe these pathologies and offer an explanation for their root cause.
3. We reevaluate Internet graph reduction work, and discover that its conclusions are largely confined to the data set its authors studied. Our work questions the generality of the results obtained therein, and offers a broader picture of Internet graph reduction.
4. Two new and novel graph reduction algorithms are introduced. These methods outperform previously studied algorithms in achieving desirable reduced graph characteristics across multiple Internet topology sources and timeframes. We observe increased performance with our reduction algorithms over a larger time period, with higher average degree initial instances.

1.3 Thesis Structure

This thesis is divided into six chapters as follows:

1. In Chapter 1, we motivate Internet measurement and topology inference, and discuss why it is a difficult undertaking.
2. In Chapter 2, we survey Internet topology inference techniques and discuss methods to perform Internet experiments in a laboratory environment. In addition, we discuss algorithms for constructing realistic Internet topologies for use in simulated or emulated environments.

3. Chapter 3 describes a new and novel tool we designed for Internet topology measurement experiments.
4. In Chapter 4, we describe our methodology for obtaining realistic Internet-like topologies, as well as introduce a novel technique for doing so.
5. Chapter 5 discusses results obtained from runs of our automated topology inference tool. We also evaluate the results of performing Internet graph reductions using our own algorithms as well as applying prior work algorithms to new data sets.
6. In Chapter 6, we detail our conclusions and possibilities for future work in this domain.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background and Related Work

In this thesis, we present a platform to perform network topology inference on emulated networks. As such, we first discuss previous work in the field of Internet topology inference in Section 2.1. This includes various methods of obtaining inferred graphs of networks and the Internet as a whole. Second, Section 2.2 motivates the choice of emulation for our experiments, rather than simulation or building physical networks. Finally, in Section 2.3, we discuss ways in which realistic, small-scale models of the Internet topology have been generated.

2.1 Topology Inference

Significant work has been performed in the field of mapping Internet topology. Research in network topology inference can be classified by two broad categories — those in which the measurements of network topology are conducted in a passive manner, and those in which the networks in question are actively probed. Within the passive topology measurement domain, administrative messages such as BGP route advertisements and withdrawals are collected. Compiling these messages over a period of time, perhaps through routers that peer with others in various ASs around the world, provides a control-plane snapshot of the Internet. The University of Oregon's RouteViews project does this on a global scale; other projects have involved gathering BGP advertisements and withdrawals from a university's border router [3].

Active topology inference involves sending data packets from *monitors*, or hosts designated for this purpose, that are attached to a network via their particular *vantage point*, or gateway to the wider Internet. Because active probes are generally constrained to follow data-plane paths dictated by the routers, additional vantage points help reveal more of the topology. The network mapping utilities that have been developed for this purpose usually build on some variant of the well-known network diagnostic tool *traceroute*. While active probing provides the benefit of being able to solicit Internet Protocol (IP)-layer path information for networks on demand, there are many problems with *traceroute*-based tools as well.

2.1.1 Limits of *traceroute*

The *traceroute* utility is a useful tool to assist in resolving network connectivity issues experienced by users and network operators. *Traceroute* works by sending a series of User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP) Echo Requests, or Transmission Control Protocol (TCP) synchronization (SYN) packets to a distant host. Initially, the IP Time-to-Live (TTL) value is set to one, and with each successive packet, the TTL is increased by one. Intermediate routers decrement the IP TTL value at each hop, and when it reaches zero, return an ICMP Time Exceeded message to the IP address that initiated the *traceroute*. Because these ICMP Time Exceeded messages contain the IP address of an interface on the router at which the packet's TTL reached zero, the forward router interface path from the *traceroute* originator to the destination IP address can be established in an ideal scenario. In reality, however, there are many ways in which the results we obtain from *traceroute* may not provide us with complete topological information.

1. *traceroute* results, even under ideal circumstances, only provide insight into the path packets take from the probe's source to the specified destination. The reverse path, that taken by the ICMP Time Exceeded replies, or data traffic from the *traceroute* destination to the machine that originated it, may be drastically different. The implication for topology inference is that a network graph obtained from a single monitor represents only a snapshot of the topology, as "seen" from that source.
2. When a router receives a packet with a TTL of one, it decrements this value and returns an ICMP Time Exceeded message to the source IP address of the original packet, as described above. However, the IP address that it responds from, or lists as the source in returned the ICMP Time Exceeded message, is often, but not always, the IP address of the interface on which it received the time-expired packet. This means that creating a router-level graph of the topology requires knowing which IP addresses correspond to the same router. There has been much work into this problem, known as *alias resolution*, as it consists of determining which IP addresses, or aliases, correspond to a single device. Techniques for resolving IP addresses to single routers include sending UDP packets with a high TTL to a high numbered port, which may elicit a "UDP port unreachable" message from a common IP if the two interfaces tested reside on the same machine [4]. More sophisticated and effective methods involve exploiting properties of the returned IP identification field [5], [6].

Regardless of the method used to resolve IP addresses to individual routers, alias resolution is an imperfect process, yet is required in addition to the IP-layer probing.

3. Because active probing methods require data to traverse the network to obtain topological information, these probes are subject to the same treatment as other network traffic. This includes router behavior designed to reduce congestion along one particular path when many, equal cost paths are available — commonly referred to as "load balancing." Load balancing is typically performed on a per-flow basis. The effects of load balancing on *traceroute* probes are well known. Consider the example topology of Figure 2.1, and the incorrect *traceroute* inference in Figure 2.2. Figure 2.2 shows topological information inferred that does not exist in reality, while also missing edges along the true topology. A specialized version of *traceroute* known as *Paris-traceroute* has been developed to mitigate the effects of per-flow router load balancing [7] by varying only UDP, ICMP and TCP header information that is not used by routers for flow determination. Nevertheless, because inference tools may not leverage this optimization, router load balancing remains a source of erroneous topological information.
4. Further, intermediate routers between the *traceroute* source and the target may be configured in such a way that no useful topological information is received from them at all. These routers, known as "anonymous routers," have been configured to ignore packets whose TTL is decremented to zero upon arrival, and do not respond with an ICMP Time Exceeded message. Instead, the packet is merely discarded. Results from a *traceroute* in which an anonymous router is encountered will display a "*" in place of that hop. For topological inference experiments, the result is a disconnected path, for we have no IP address information for the hops at which an anonymous router was discovered.
5. Finally, many networks filter traffic in order to reveal as little as possible about their own topology to outsiders. This tactic is often employed for security purposes; without a topological understanding of a target network, attackers may find it more difficult to exploit weak or important devices within the network. In terms of topology inference experimentation, the result is that *traceroute* or other inference tools may not be able to return results beyond the "edge", or Point-of-Presence (PoP), at which inference traffic begins to be filtered.

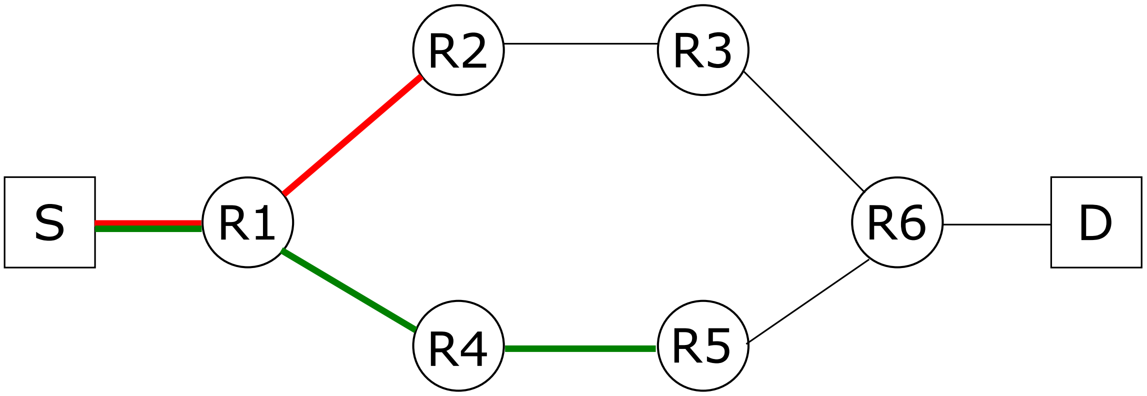


Figure 2.1: A *traceroute* from S to D. The packet with TTL=2 traverses the upper path, which is denoted in red. The packet with TTL=3 traverses the lower path due to load balancing at R1, and is shown in green. The *traceroute* results would indicate a link connecting R2 to R5 when in fact none exists.

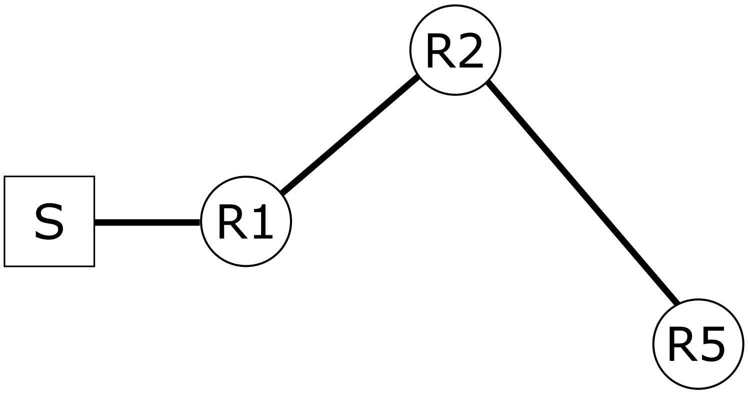


Figure 2.2: The path through TTL=3 as seen by the *traceroute* initiator S.

2.1.2 Non-*traceroute*-based inference tools

Before describing current *traceroute*-based topology inference tools, we begin our discussion with MEasure the Router Level of the INternet (MERLIN), which instead exploited a control-plane management tool to obtain topological data.

MERLIN [8], used the multicast management utility *mrinfo* in order to infer topology. Multicast-enabled Internet Protocol Version 4 (IPv4) routers, when queried by *mrinfo*, would return results about neighboring multicast routers connected via the interfaces of the router being queried. In a similar manner, *mrinfo* could then be used to query the neighboring multicast routers to obtain their multicast neighbors, and so on. When applied exhaustively, a complete topological picture of the multicast-enabled network could be obtained. Further, because *mrinfo* identified unique routers, it had the distinct advantage of not requiring alias resolution, unlike *traceroute*-based topological discovery techniques. Unfortunately for topology inference purposes, multicast routing is no longer widely used on the Internet. Its disuse is mainly attributable to a lack of economic incentive for companies operating multicast routers to replicate data for non-customers. Further, multicast routing can cause potentially devastating effects to networks called "multicast routing storms" when not configured correctly. For these reasons, the methodology used in the MERLIN project does not provide a feasible means to obtain topological data any longer.

2.1.3 *Traceroute*-based topology inference

Unlike MERLIN, *skitter* and its successor *scamper* are *traceroute*-based software probing tools for active topology inference. Because *scamper* fully replaced *skitter* in large-scale Internet mapping projects, we will focus on it here exclusively.

Scamper [9] is a parallel packet prober for topology inference. *Scamper* supports UDP, UDP-paris, ICMP, ICMP-paris, and TCP probing methods, and will fill a per-second packet probing rate specified by the user with probes. Like *traceroute*, *scamper* sends a series of probe packets with increasing TTL values in order to elicit ICMP Time Exceeded responses from intermediate routers along the path to a destination IP address. In order to combat router load balancing along the path, the UDP-paris and ICMP-paris methods employ the same mechanics as *Paris-traceroute* to allow every probe to traverse the same load balanced path. The parallelizability of *scamper* makes it ideal for distributed topology inference; an individual or organization with monitors located in disparate locations can simply run *scamper* from these vantage points, and *scamper* will return when it has finished collecting results. Finally, *scamper* writes probe response data in an compressed, binary format called *warts*. Compressed result data are a requirement for large-scale inference projects in which millions of destinations are probed, eliciting tens of millions of intermediate hops along the way.

Organizations such as Center for Applied Internet Data Analysis (CAIDA) and Réseaux IP Européens (RIPE) continuously probe and collect inferred snapshots of the global Internet topology for both IPv4 and Internet Protocol Version 6 (IPv6). CAIDA's Archipelago (Ark) and RIPE's Atlas networks consist of more than a hundred and thousands of monitor nodes, respectively, located around the world. These monitors provide locations from which to perform network measurement experiments, primarily using the *ping*, *traceroute*, and *scamper* utilities as well as Domain Name System (DNS) record lookups. Of interest to our study are the inferred topology graphs of the Internet created through repeated cycles of probing from the monitor nodes run by CAIDA and RIPE.

Using *scamper*, CAIDA probes to an IPv4 address in each of the /24 prefixes of the routable IPv4 space once approximately every 48 hours from their Ark monitors. The union of these probe results forms a picture of the Internet during this two-day snapshot in time, which can be considered a graph whose vertex-set consists of every IP address discovered and whose edges are logical connections between them.

Unfortunately, there are many limitations with this inferred data, including those discussed in Section 2.1.1. First, *scamper* relies on sending a series of UDP, TCP or ICMP packets to elicit ICMP Time Exceeded responses from IP-layer devices. These probe packets are naturally constrained by the data path; that is, they are forwarded from router to router by the information contained in these devices' Forwarding Information Bases (FIBs). Forwarding information is often determined by the shortest path through the network, but network operators can also inject policy and design choices into the path selection logic. Those initiating a trace via *scamper* or *traceroute* have no control over the path selection these intermediate routers make once the probe packets have been sent into the network; the routing determination is entirely up to the algorithms and policy in place on these devices. Exacerbating the problem, path selection may change over time due to design or policy changes. Further, the vantage point from which active measurement probing is sourced limits our inference: network policy may not allow probes to travel over certain physical paths due to the probe's source IP address, or certain paths may never be selected because they are deemed less desirable by the routers along the data path. A trivial example of the differences in inferred topologies is illustrated in Figure 2.3. The monitor inferring the three-router topology connected to Router 1 does not discover the link between Routers 2 and 3 due to router path selection, while the inferred topology from Router 2 discovers the link between Routers 2 and 3 but misses the link between Routers 1 and 3. While in practice route selection is much more complicated, Figure 2.3 serves to motivate the problem. Additionally, link-

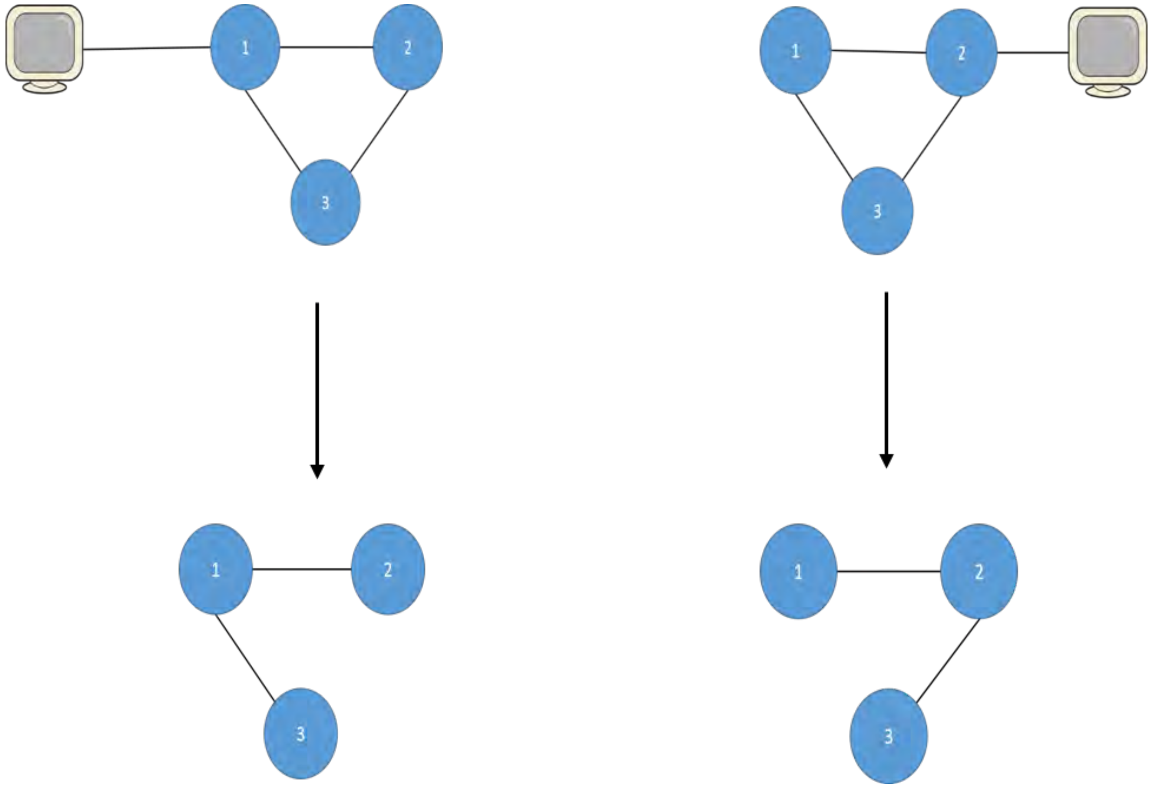


Figure 2.3: Two identical router topologies – one with a monitor using Router 1 as a vantage point, the other using Router 2 as a vantage point. The inferred topology on the left captures the link between Routers 1 and 3 but misses the link between Routers 2 and 3. Conversely, the inferred topology on the right captures the link between Routers 2 and 3 but misses the link between Routers 1 and 3.

layer topology may affect the inferred topology at the IP layer [10]. Figures 2.4 and 2.5 depict classic examples of these effects. Figure 2.4 shows a true topology, consisting of one central switch connected in a star configuration to four routers. Figure 2.5, however, displays the inferred topology using a IP layer tool, like *scamper*. Our inferred topology not only discovers two additional links that are not present physically, but it also conveys a much more robust architecture than is actually present.

Finally, we do not have underlying ground truth available for most networks, and certainly not for the Internet. Thus, we have no way to ascertain the accuracy of inferred network topologies, including the complete Internet graph. The unavailability of ground truth occurs for several reasons. First, network diagrams and policy decisions are often a closely-held secret for network operators and corporations. This is partly due to security concerns; as mentioned previously, an attacker with an understanding of a network's structure can focus his resources on attacking the most vulnerable points to maximize the effects of his attack. Economic interests also motivate decisions to keep network topology confidential. One can imagine an ISP that has designed its network to maximize throughput and minimize latency for its customers beyond the that of its competitors. Guarding this trade secret, rather than allowing rivals to duplicate its superior service, is in the ISP's best financial interest. Finally, IP networks are dynamic; they respond to physical disconnections or hardware failures by rerouting traffic, adapt to increased or decreased load on connections, and are subject to the policy implemented by their operators, which can be modified at will. For these reasons, any topological snapshot we obtain is almost certainly incomplete.

Because obtaining a complete picture of the underlying ground truth of real networks may be impossible, we endeavor in this thesis to create our own ground truth for the purpose of evaluating the results obtained by network measurement tools. In Section 2.2, we briefly consider previous work in creating networks that may be used for this or similar purposes.

2.2 Real, Simulated, and Emulated Networks

Operators and researchers are interested in performing experimentation and testing on networks that closely or exactly model network topologies of interest. For example, engineers working for an ISP may wish to study the effects that a change in policy may have on their topology's resiliency or latency. In order to guarantee that an acceptable quality of service will continue uninterrupted to its customers, sets of link and router outages could be studied to quantify the impact of these failures with respect to network capacity and latency. Further, so-called "flash crowds" are a phenomenon wherein a network or web site experiences a sudden, unexpected surge in traffic [11]. Content Distribution Networks (CDNs) operators and ISPs may wish to understand the volume of traffic they are capable of handling, and take steps to mitigate weaknesses before experiencing a flash crowd event. With the results of these experiments, design decisions could be rethought, or policy changes implemented. In addition to the commercial benefits of modeling network topologies, researchers may want to test the performance of a new protocol on a realistic, "Internet-like" network.

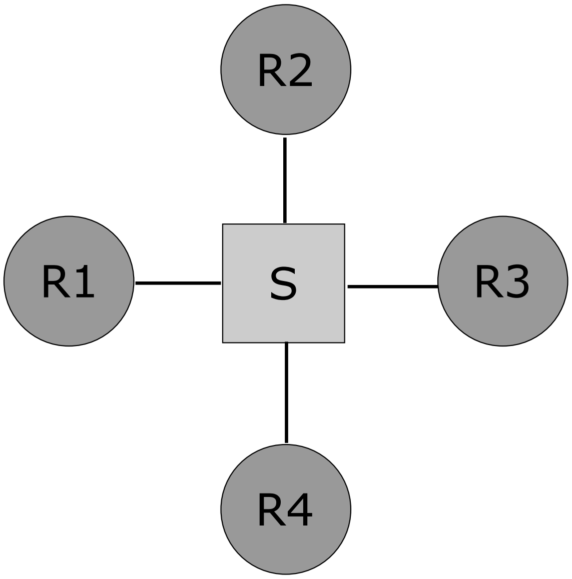


Figure 2.4: A network consisting of one central switch connected to four routers. The presence of the switch, a link-layer device, in this topology can cause misleading inference results when probed by an IP-layer tool like *scamper*.

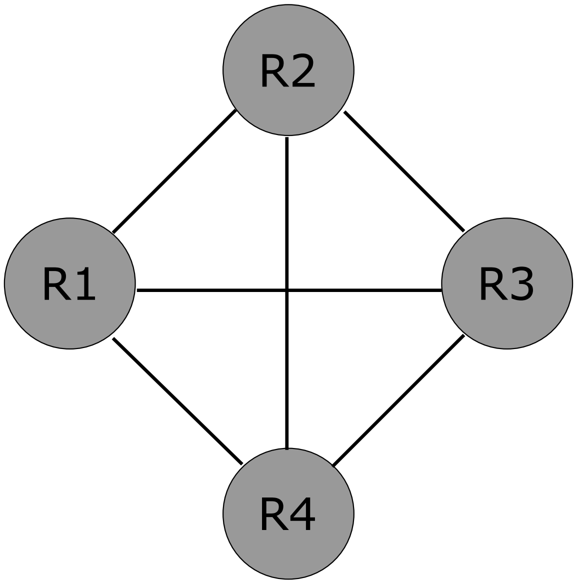


Figure 2.5: The inferred topology of the single switch, four router network to the left, obtained by network mapping tools that operate at the IP-layer.

Regardless of their motivation, many individuals have a need for large-scale network topologies for testing and evaluation. Building topologies of non-trivial size in physical hardware can be cost- and time-prohibitive. Even small to medium sized networks consist of hundreds to thousands of routing and switching devices. In addition to the obstacle of obtaining large quantities of routing hardware, hours or days of time can be necessary to physically cable and configure the desired topology - an error-prone process in itself. And, each new topology incurs this reconfiguration cost.

On the other end of the spectrum from physical networks, simulation provides an alternative that is far less expensive and time-consuming. Network simulation software, like the open-source *NS3* [12], the commercial product *OPNET* [13], and Cisco's *PacketTracer* and *NetSim*, fulfill the need to perform experimentation and testing on artificial networks in an efficient, cost-effective way.

These tools, however, carry a significant disadvantage as well. Simulation software tools abstract away the underlying processes and mechanisms as actually implemented by the hardware and software systems they are simulating. For example, rather than running the underlying operating system (OS) of a router, a network simulation program may actually simply idealize the routing protocols the user wishes to observe. In effect, network simulation loses the real-world implementation specific details that are inherent in a physical network in exchange for a time- and cost-effective model of a network topology's characteristics.

A third option, emulation, seeks to find middle ground between the extremes of network modeling through physical hardware and pure simulation. In emulation, a network device, such as a router, switch, or host, is run in a virtual environment. Further, the device's real software OS is emulated, i.e., it runs in this virtualized environment. Leveraging emulated, virtualized network topologies provides the advantages of both building physical networks and simulating networks. The emulated OS provides implementation-specific details, defects, and bugs that are present in hardware networks, while virtualization allows the set-up and configuration of many devices on a single host.

Graphical Network Simulator - 3 (GNS3) is one such emulation platform. GNS3 presents a graphical interface through which users can build network topologies by dragging and dropping network devices onto a canvas, then interconnecting them through virtual linkages. Underlying GNS3 is the software program *Dynamips*, which is an emulation platform for Microprocessor without Interlocked Pipeline Stages (MIPS) instruction set Cisco processors. GNS3 is widely used to build small-scale network topologies for instructional purposes. *Netkit* is another emulation product [14]. *Netkit* works by creating a number of emulated User-Mode Linux (UML) kernel devices, upon which other software is run to emulate a particular device. The open-source *Quagga* software, with its associated daemons for various routing protocols, could be installed to emulate a router, for example. *AutoNetkit* allows for the construction of networks from simple topology diagrams created in *yEd*, an open-source network visualization tool. Additionally, *AutoNetkit* provides a high-level programming language extension to Python to perform administrative tasks such as forming collections of routers into ASs, creating routing process groups, and device configuration. When the network has been configured, *AutoNetkit* provides a web-based user interface (UI) to visualize the network and view the results of measurement tools, namely *traceroute*. The program then provides output that can be used by emulation platforms like

Netkit for experimentation and testing [15], [16], [17]. *AutoNetkit* has recently been integrated into Cisco's *Virtual Internet Routing Lab (VIRL)* software, a competitor of GNS3, to assist in quickly performing the administrative and configuration functions described above. [18]

2.3 Building Realistic Topologies

In order to obtain realistic, Internet-like topologies, two distinct types of methodologies have been proposed. The first is a constructive approach — beginning with a small graph, then adding nodes and edges until an instance of the desired vertex-set cardinality has been reached. Many methods have been proposed to accomplish this, including the Barabási-Albert, Waxman, Inet, and LocGen generation models [19], [20], [21], [22]. Each is described briefly below.

2.3.1 Barabási-Albert Generation Model

The Barabási-Albert (BA) generation model is a naïve approach to creating Internet-like topologies, but it is well-known for its ability to create networks with a power law degree distribution [19]. Following the proposal of this incremental growth, preferential attachment network generation model, many biological, social and technological networks were discovered to exhibit the power law degree distribution property of BA networks. Based on data from the University of Oregon's RouteViews project, Faloutsos *et al.* posited that the Internet's AS-level graph exhibits the power law properties that would be expected from BA model generation [23]. However, this study was repudiated in [24], using a more complete data set that included RIPE and AS *Looking Glass* BGP information in addition to the RouteViews BGP data in the original paper. Further, while BA-generated topologies produce high-degree "hub" nodes, debate over their relationships to each other in the Internet exists due to the incomplete picture we have from topological mapping [25]. Despite these criticisms, we present BA-generated networks as a simplistic approach to Internet topology generation.

Given an initial connected graph of k nodes, new nodes are added to the network individually, and preferentially form connections with a node n_i already in the network with a probability of

$$p_i = \frac{deg(n_i)}{\Sigma_j deg(n_j)}$$

for $0 \leq j \leq k-1$.

2.3.2 Waxman Generation Model

The Waxman network generation model is useful for its ability to model the principle of locality that exists within networks [20]. Because network operators are constrained by cost and physical location, they are more likely to connect to the closest service provider that has sufficient bandwidth or charges them the least, rather than selecting a provider to connect to that is geographically remote.

With all points positioned in the plane, either randomly or by a heavy-tailed Pareto distribution, the Waxman model assigns the probability of an edge connecting nodes u, v by

$$p(u,v) = \alpha e^{\frac{-d}{\beta L}}$$

where $0 < \alpha, \beta \leq 1$, d is the Euclidean distance between u and v, and L is the maximum Euclidean distance between any pair of nodes on the plane. Waxman topologies, while excellent at modeling the propensity for links to be established by entities in close physical proximity to each other, fail to capture the extremely high degree of “hub” nodes observed within the actual Internet [21].

2.3.3 Inet Topology Generator

The Inet Topology Generator has undergone several iterations, most recently Inet-3.0 [21]. Inet is a Unix-based software program that takes user-defined parameters, and outputs a text file containing an AS-level Internet topology. The parameters Inet defines for user-specification are $N \geq 3037$, the number of nodes desired in the topology, d, the fraction of 1-degree nodes in the topology, p, the size of the plane the topology should be built in, and s, a random seed. The text output specifies the degree of each AS, each ASs position in the plane, and each inter-AS link as a pair.

Inet topology generation begins by placing the N nodes in a plane, and assigning an out-degree to the top 2% of nodes based on the equation

$$deg_{out} = e^{pt+q} r^{R}$$

where p, q, and R are constant, e is Euler's number, r is the rank of the vertex, and t is the number of months that must have elapsed since November 1997 for an Internet instance to have reached N nodes based on an exponential growth assumption. Following this, the p percent of nodes the user specified as degree 1 are designated, and lastly, the remaining vertices in the graph are assigned out-degrees based on the above equation.

The remaining steps of Inet are focused on producing a connected network, the details of which we will not concern ourselves with here. More toward our purpose, Inet topologies claim to follow the Internet's characteristics better than other constructive models at node numbers studied in [21], namely 6,000. This graph roughly corresponds to the Internet's vertex-set cardinality in October 1999. Inet's superior performance relative to other models belies the fact that its highest degree AS has an out-degree of approximately half that of the Internet instance, and the authors themselves admit their model follows the spectral behavior of the Internet graph poorly [21].

2.3.4 LocGen Topology Generation Model

Recognizing that cost and physical location of topology generation models are considered secondarily or not at all, researchers at the University of Kansas endeavored to create a model that accounts for these two properties [22]. Using three random link generation models, *pure random, locality*, and Waxman, the authors incorporate a cost variable that includes a fixed cost, plus a variable cost per unit of distance between two nodes, which is multiplied by the Euclidean distance separating the two. Given node placements from the Sprint router-level network as inferred by the *Rocketfuel* project [26], and a PoP-level network of the European research network GEANT2, the authors compare graphs created by the pure random, locality-based, and Waxman models to those generated by the same models, with inclusion of the cost variable described above.

The results of comparing these generated graphs to the actual topologies of the Sprint and GEANT2 networks is favorable toward their cost-inclusive models in the metrics of degree distribution and average shortest path length. In particular, the cost-inclusive Waxman model performed particularly well given the node-sets for these two topologies. With the original Sprint topology, the cost-inclusive Waxman model generated the same average degree of 5.04, and an average shortest path that differed by only 0.07 to the real network. Similar performance was seen when compared to the GEANT2 network; only a 0.06 difference in average degree and 0.13 difference in average shortest path length was observed.

2.3.5 Hierarchical Internet Model

For the majority of our experimentation, however, we took a simplistic approach to generating hierarchical Internet-like graphs. These tiered models attempt to generally replicate the high-level structure of the Internet by dividing each router, representing an AS, into one of three categories - Tier 1 backbone, Tier 2 transit provider, or customer. Tier 1 ASs, as in the actual Internet, are few in number and represent the core of the network in our model. Tier 2 ASs are more numerous and are analogous to country or regional service providers on the actual Internet, while customer ASs form the majority and represent enterprise networks. For this hierarchical model, input parameters include the number of ASs desired, and the fraction to be Tier 1, Tier 2, and customer ASs. Further, a Tier 2 AS peering probability must be specified — this parameter determines the fraction of Tier 2 ASs that will form settlement-free links to each other to transit traffic without first traversing the Tier 1 backbone. Further, the probability with which customer ASs will connect to multiple Tier 2 providers (known as *dual-homed*), and the probability that Tier 2 providers will form one, two or three customer-provider links with Tier 1 ASs are specified. Our final parameter is a Boolean value indicating whether link failure scenarios will be examined for this topology. In Section 3.2.1, we will visit this tiered model for network generation again.

2.3.6 Graph Reduction Methods

A second approach topology generation is a reductive one — beginning with an initial Internet instance, and reducing it until a graph of the desired vertex-set order is obtained, retaining the properties that an Internet instance *of the reduced order* would have. This approach was studied in detail in [27], and we undertake a reexamination of the graph reduction methods introduced by the authors in order to evaluate their viability with current

Internet data. From these reductive methods, we explore our own, and choose the best to reduce topologies to a node-set cardinality able to be emulated on our platform. We present an overview of the work of [27] here.

Using data publicly available from the RouteViews BGP RIB data set, prior work investigated the possibility of using known Internet graph instances to reduce to graph instances of a specified node-set cardinality. In order to evaluate the effectiveness of these methods in creating Internet like graphs, the authors of [27] begin with an Internet instance from May 2001, reducing it to the node-set cardinality of an Internet instance inferred in January 1998. To quantify how well their reduction methods perform, the prior work authors chose to evaluate the similarity of the method-reduced graphs to the Internet in January 1998 in terms of average degree, spectral behavior, hop-plot, and power-law exponents.

Prior work has identified three different categories of graph reduction, namely *contraction*, *deletion*, and *exploration*. Each provides a different avenue to reduce a large graph to one of smaller order, and are briefly described here.

Contraction Methods

We examine two different types of contraction methods used in [27]. In *Contraction of a Random Edge (CRE)*, an edge in the graph is selected with a uniform probability. The vertices incident to this edge are contracted, or merged into a single vertex, which now has the union of edges incident to the original two vertices incident to itself (minus the edge that has been contracted). We demonstrate CRE in Figure 2.6.

The *Contraction of a Random Vertex-Edge (CRVE)* [27] method first selects a vertex in the graph with a uniform probability, then chooses an edge incident to that vertex with a uniform probability. The endpoints of this edge are then contracted into a single vertex, which has incident to itself the union of edges incident to the original two vertices minus the edge that has been contracted. A small example of CRVE is given in Figure 2.7.

In both CRE and CRVE, the contraction process is repeated until a graph of the order of the reduction target is obtained.

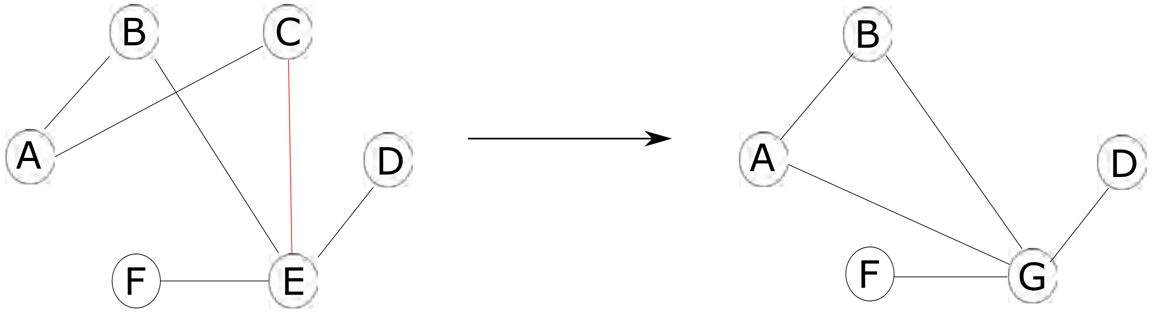


Figure 2.6: A contraction by CRE. First, the edge (C,E) is randomly selected. Next, vertices C and E are contracted into a single vertex, labeled G. Vertex G retains all edges that were incident to the C and E, minus the edge (C,E), which has been removed.

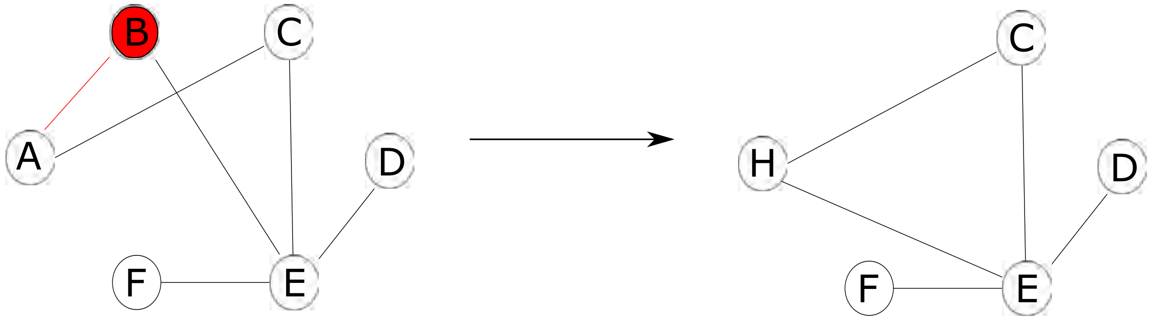


Figure 2.7: A reduction by the CRVE method. First, vertex B is randomly chosen. From the edges incident to B, the edge (B,A) is randomly selected, and vertices B and A are contracted into a new vertex H. H retains all edges that were incident to A and B, with the exception of (B,A), which has been removed from the graph.

Deletion Methods

Within the deletion category, we consider three distinct methods and a hybrid of two of these methods. The first method, *Deletion of a Random Vertex (DRV)* [27], involves selecting a random vertex with a uniform probability, then removes this vertex (and its incident edges) from the graph. After each vertex removal, a check is performed to ensure that resulting graph is still connected. If a disconnected graph is produced, the largest connected component is retained, while all other components are removed. This process repeats until the method produces a reduced graph of approximate order to the Internet instance to which we are reducing. Equality of order between the reduced graph and the reduction endpoint may not be realized, as a disconnection and the subsequent removal of disconnected components may reduce the graph beyond the number of nodes desired. However, by checking for disconnections after each vertex removal, we obtain a comparable number of vertices in the reduced graph to the final Internet instance. A graphical example of DRV is given in Figure 2.8.

Second, the *Deletion of a Random Vertex-Edge (DRVE)* [27] method begins by first choosing a vertex at random from the Internet instance with a uniform probability. Then, an edge incident to this vertex is selected with a uniform probability, and removed from the graph. Again, the method checks for disconnections following each edge removal, and discards all but the largest connected component of the resultant graph. This process repeats until it produces a reduced graph of comparable order to the target instance. Figure 2.9 demonstrates a reduction by DRVE.

The third method, *Deletion of a Random Edge (DRE)* [27], selects a random edge with uniform probability, and removes it from the graph. As in DRV and DRVE, following the removal of each edge, all but the largest connected component are removed if a disconnection results from the edge removal. This process repeats until it reaches the limit of our reduction. An example of one edge deletion in DRE is given in Figure 2.10.

Finally, we examine a set of *Deletion Hybrid (DHYB)* [27] methods, which are hybrids of DRE and DRVE. Specifically, each DHYB-p method chooses DRVE with a probability of p and DRE with a probability of $(1-p)$. Thus, when $p = 1$, this is equivalent to DRVE, and when $0 = p$, is equivalent to DRE. In our work, we examine values of p between 0 and 1 in one-tenth intervals.

Exploration Methods

We consider two distinct methods within the exploration category - *Exploration by Depth-First Search (EDFS)* and *Exploration by Breadth-First Search (EBFS)* [27]. In EDFS, a vertex in the initial Internet instance is selected randomly with a uniform probability. From that vertex, the Depth-First Search algorithm [28] is run until the desired number of vertices have been discovered. The result is the induced subgraph of the original Internet instance induced by the node set of the Depth-First Search tree. A small example of an 8-vertex graph reduced to a 5-vertex graph by EDFS is given in Figure 2.11.

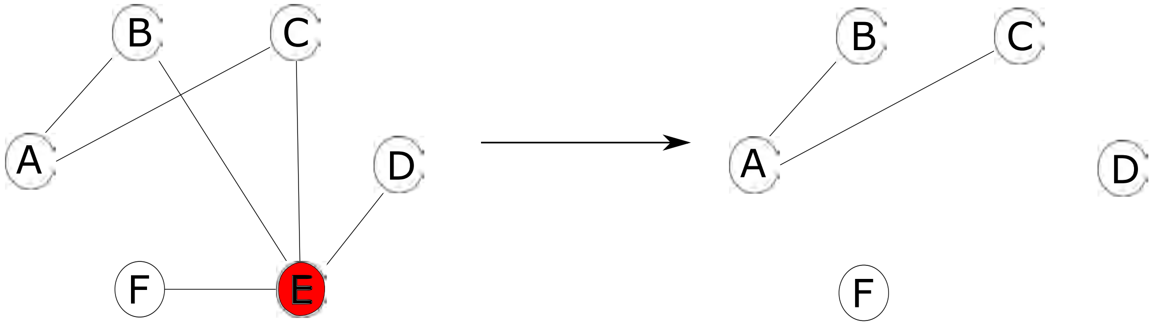


Figure 2.8: A reduction by the DRV method. Vertex E is randomly selected from the node-set and removed from the graph with its incident edges. A check for connectedness follows, at which point vertices D and F will also be removed, leaving only the largest connected component consisting of A, B, and C.

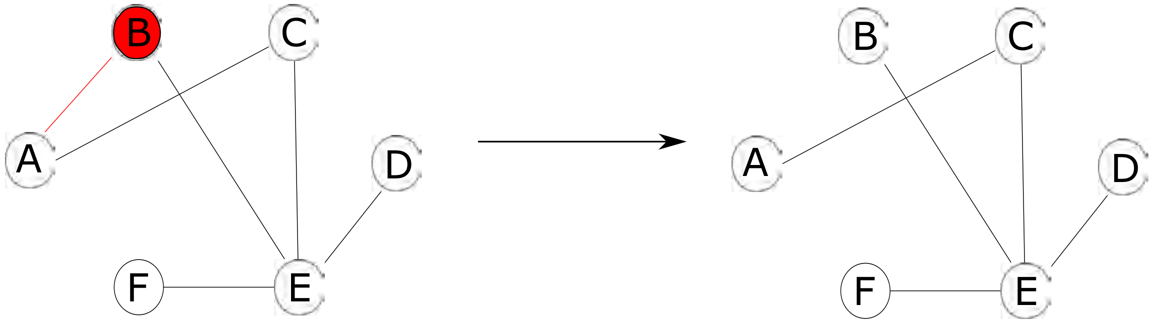


Figure 2.9: A reduction by the DRVE method. First, a random vertex from the node-set is selected, in this case, B. Next, the edge (B,A) is randomly chosen from the edges incident to B, and removed from the graph. A check is done for graph connectedness, and only the largest connected component is retained. In this example, the graph remains connected after edge deletion.

In a similar fashion, EBFS begins by selecting a random start vertex uniformly, and running the Breadth-First Search algorithm [28] until the number of vertices desired has been reached. The node set of the Breadth-First Search tree is then used to induce a subgraph of the original Internet instance, which is the resultant graph obtained. Another small example of an 8-vertex graph reduced to a 6-vertex graph is given in Figure 2.12.

Table 2.1 summarizes the reduction methods considered by prior work for quick reference.

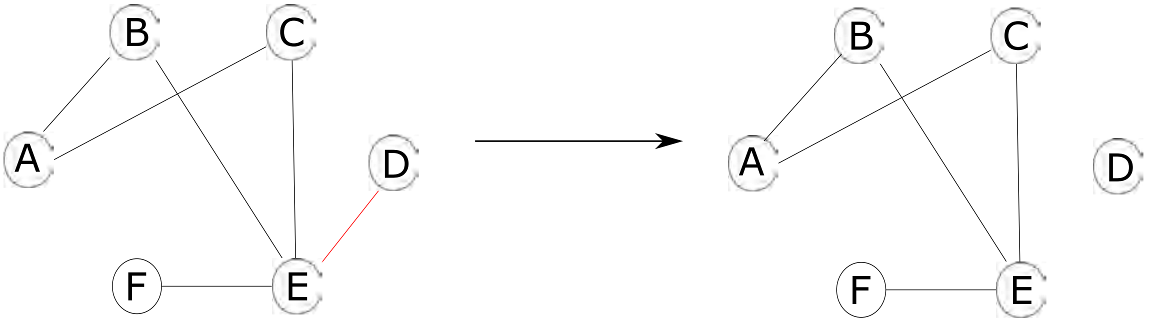


Figure 2.10: A reduction by the DRE method. Edge (D,E) is randomly selected from the edge-set, and removed. Following edge removal, a check for connectedness is performed, and only the largest connected component is retained. In this case, the isolated vertex D will also be removed.

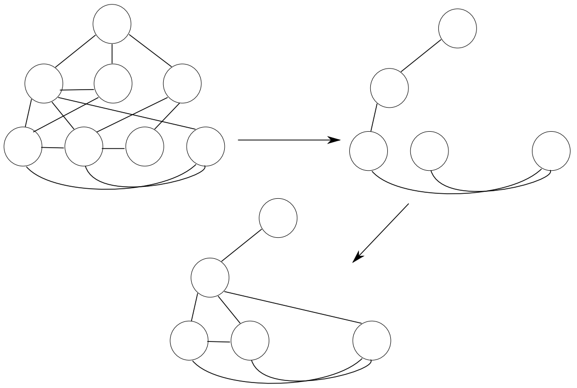


Figure 2.11: A reduction by the EDFS method. Beginning with the top-most vertex, Depth-First Search is performed until a specified number of nodes have been visited, in this case, 5. The subgraph of the original graph induced by the visited nodes is retained.

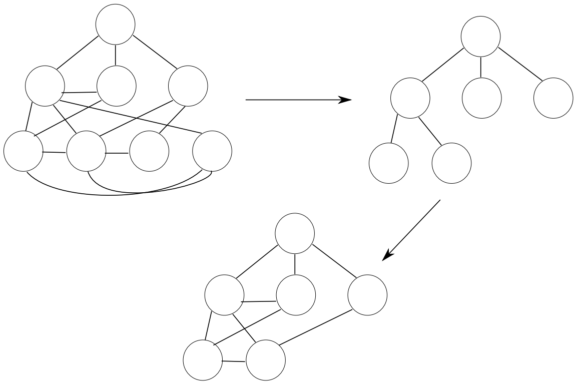


Figure 2.12: A reduction by the EBFS method. Beginning with the top-most vertex, a Breadth-First Search is executed until the desired number of vertices have been visited. In this case, the specified number is 6. The subgraph of the original induced by the visited nodes is retained.

Prior Work Results

The authors of [27] found, based on an examination of several graph metrics, that DHYB-0.8 outperforms all other reduction methods. Metrics of interest to the authors were the average degrees of the reduced graphs over time — from the original instance to the reduction end point, the spectral behavior of their reduced graphs compared to the Internet instance from January 1998, and *hop-plot*, a cumulative function counting the fraction of vertex pairs within k hops or less of each other along a geodesic of the most reduced graphs. The data set studied in [27], however, was of limited scope, covering only RouteViews BGP

Graph Reduction Methods		
Abbreviation	***Name***	***Description***
CRE	Contraction of a Random Edge	A random edge is selected with uniform probability. Adjacent nodes to that edge are contracted into one.
CRVE	Contraction of a Random Vertex-Edge	A random vertex is selected, then a random edge incident to that vertex. Adjacent nodes contracted.
DRE	Deletion of a Random Edge	A random edge is selected with uniform probability, then deleted from the graph. If disconnections occur, keep largest connected subgraph.
DRV	Deletion of a Random Vertex	A vertex is selected with uniform probability, then deleted from the graph. If disconnections occur, keep largest connected subgraph.
DRVE	Deletion of a Random Vertex-Edge	A vertex is selected randomly, then an edge incident to that vertex is selected randomly. This edge is then deleted. If disconnections occur, keep largest connected subgraph.
DHYB	Deletion Hybrid	DHYB-X chooses between DRVE with probability X and DRE with probability (1-X)
EBFS	Exploration by Breadth-First Search	A vertex in the graph randomly selected as the root node. Breadth-First Search algorithm is run until desired number of nodes visited; these nodes used to induce subgraph of original.
EDFS	Exploration by Depth-First Search	A vertex in the graph randomly selected as the root node. Depth-First Search algorithm is run until desired number of nodes visited; these nodes used to induce subgraph of original.

Table 2.1: A summary of prior work graph reduction algorithms

RIB data from 1998 to 2001. Because of this, we view the results obtained with some trepidation unless proven to be consistent across multiple data sources and timeframes.

In this thesis, we endeavor to expand upon the body of work in this field by considering additional sources of Internet AS-level topology information, and by reducing more recent Internet instances to see whether they provide similar findings.

CHAPTER 3: Automated Topology Inference Methodology

Recognizing the limitations and difficulties inherent in topology inference discussed in Section 2.1, we endeavor to create our own "ground truth" in an emulated environment. Specifically, by ground truth we mean several things. First, we know of each router in the network, its interfaces, and the links that interconnect them. IP address allocation to the routers interfaces are known, as well the subnetworks each router advertises. Further, we know the routing protocols employed on the devices in the network, and the policies these routing protocols are designed to implement. Of particular importance in this thesis, we are aware of which links will fail during inference testing, and when they will fail.

In summary, our aims are to:

1. Automate the construction of ground truth, according to one of several known topology generation models.
2. Automate the set-up of the emulated environment, leveraging existing emulation software if possible.
3. Emulate a non-trivial number of routers, preferably on the order of several hundred, with non-trivial connections between them.
4. Automate topology inference. Our platform should run the state-of-the-art inference tool, *scamper*, to infer the path to an IP address on every router in the topology from a virtual monitor.
5. Further, our emulation platform should use every router in the topology as a vantage point, moving from one to the next exhaustively without human interaction.
6. Our platform should be able to simulate real-world conditions, such as link or router failures, and record the inferred topology under these event sets.
7. Lastly, our platform should record all results for later analysis.

Toward this end, we develop a tool for emulated network topology inference called *Emulated Router Inference Kit (ERIK).* ERIK is a Python-based program that accomplishes the tasks outlined above, and we believe is a useful resource for network research. We first outline the early stages of ERIK development, including our failed attempts at modifying existing emulation software for our purposes. We believe it instructive to motivate the need for developing our own software solution. Following that, we detail the current state of ERIK, and lastly, introduce two of our own methods for reducing Internet graphs for the purpose of emulation on ERIK.

3.1 Limitations and Challenges Experienced in Developing ERIK

Our first attempts at automatically building emulated networks for research in network inference centered around GNS3, the graphical network emulator described in Chapter 2. GNS3 topologies are simple to automatically construct via a script - they consist of a set of Cisco router configuration files (one per router) and a "*.net*" topology file, which contains metadata about the network. In particular, the *.net* file holds information pertaining to the position of each router on the GNS3 graphical plane, the TCP port to connect to each router on, and on which *Dynamips* hypervisor each router will run.

When we had reverse-engineered the parameters needed from the topology file to construct our own GNS3 networks, we first selected Routing Information Protocol (RIP) as the routing protocol we would use throughout our topology. Though RIP is no longer widely deployed, its simplicity allowed us to focus less on routing protocol-specific issues and more on network creation. In order to automatically generate these topologies in GNS3, we first construct a graph according to the Barabási-Albert, Waxman, or Erdős-Rényi Random graph generation models via the Python network package *NetworkX* [29].

From our *NetworkX* graph, it is a relatively minor task to create the router configuration files for a pure RIP topology. ERIK simply iterates through the edges of our generated graph, assigning IP addresses to available router interfaces and adding each router's networks to its RIP configuration. For our purposes, IP addresses were allocated out of a $/24$ network, and each router to router link was given a $/30$ subnet.

After the router configuration files are generated and written to disk, our next task was to generate the GNS3 *.net* topology file. This consists of allocating a TCP port per router to communicate with our emulated routers, positioning each router on the GNS3 graphical

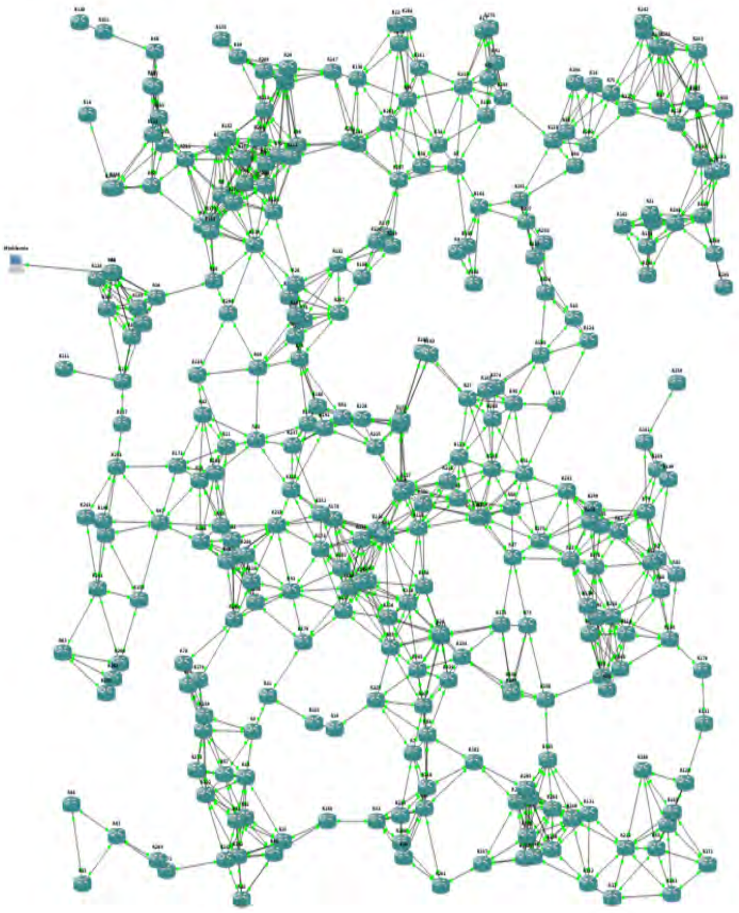


Figure 3.1: A 300-router GNS3 topology automatically generated by an early ERIK prototype.

plane, specifying which Cisco router Internetwork Operating System (IOS) to use for each router, and allocating routers to Dynamips hypervisors for emulation. Further, each router contains a list of its connections to other routers in the topology. Once this process was completed, our topology file was written to disk, completing a valid GNS3 topology. Figure 3.1 depicts a 300-router GNS3 topology created via the automated process described above.

After this proof-of-concept, the next logical step was to create a topology using a more current, robust routing protocol like Open Shortest-Path First (OSPF). This change necessitated only minor changes to the router configuration files - in place of the RIP configuration commands, the analogous OSPF configurations are written. For simplicity, we chose to build only one OSPF area, although one could foreseeably create different OSPF areas by grouping routers by Euclidean distance, given x-y coordinates for each, without much difficulty.

In both RIP and OSPF topologies, we tested our emulated topology generation with Barabási-Albert, Waxman, or Erdős-Rényi Random network models initially on a small scale - 10 to 30 routers on a relatively low-performance, commodity server. When we were confident that our topology generation scheme was working as expected, we scaled up to the order of several hundred routers on a host with more memory and computing resources. At this point, we began to experience issues using GNS3 as a platform for ERIK. Initializing our topologies in GNS3 at our target scale took far longer than we thought reasonable - approximately an hour was spent waiting for every router in the 300-router topology shown in Figure 3.1 to boot on our emulation host, for example. During this hour, GNS3 allocated memory for the routers to be emulated, initialized hypervisor instances, created the virtual interfaces for the router-to-router connections, and graphically displayed the progress of the topology's startup.

Nevertheless, GNS3 was viewed as the most suitable candidate for creating an emulation platform for network inference testing at the time, so we next moved on to implementing the topology inference module for ERIK.

A useful feature of GNS3, and a reason for initially choosing it as our platform, is its ability to interact with virtual machines over a network bridge, or `tap`, interface. For ERIK, we intended to use a *VirtualBox* host to perform our topology inference probing. *VirtualBox* is an Oracle product for virtualizing server and desktop OSs, and for our purposes, would be the virtual environment in which our inference monitor would run. For our initial inference testing, we created a minimal Linux Ubuntu virtual machine (VM) with *scamper* installed for use as our monitor platform. This *VirtualBox* VM was then imported using GNS3's appliance import function. Once imported, our VM was connected to a predetermined router within the topology that had been selected during the topology generation phase to be the monitor's vantage point, and a text file containing the IP addresses to probe was copied to our VM. At this point, all of our infrastructure was in place and actual inference testing could begin. Testing began by manually running *scamper* from the command line of our Ubuntu VM, passing our IP address file in as an argument.

Despite our initial reservations about the boot-up time for our large GNS3 topologies, we were able to successfully conduct inference tests in this manner from a single vantage point. Our *scamper* results were copied from the VM monitor and analyzed following completion. In accordance with our stated goals for ERIK, however, we recognized the following limitations from our platform in its current state:

1. First, the overhead incurred using GNS3 was too significant during the router boot-up phase. We recognized that at least some of this overhead was incurred from tasks that mattered little to our project, namely the graphical UI that GNS3 provides.
2. Second, while we could easily automate the creation of GNS3 topologies, later including the placement and connection of the monitor VM to its vantage point, we had no easy way to script vantage point changes. This was important to our work because we wanted to be able to infer the network topology from a larger-than-singular subset of the routers, including using every router as a vantage point.
3. Lastly, this platform required too much human interaction. In addition to our difficulties in scripting a solution to change the VM monitor's vantage point, we were forced to manually begin the topology inference tests by running *scamper* from the command line of the VM. To be a valuable tool for research, we felt that this process should be automated as well. In addition, simulating real-world scenarios like link or router failures required too much user input. While GNS3 has the ability to remove links or shut down routers using mouse clicks to affect these changes, this process becomes exponentially more difficult as the number of routers and router-to-router connections grows beyond a non-trivial number. Even locating a particular device on the plane becomes a time-consuming effort.

3.1.1 Dynamips Solution

As discussed in Section 2.2, *Dynamips* is the underlying emulation software with which GNS3 interfaces. Following our assessment of the capability of GNS3 to serve our purposes outlined at the start of this chapter, we instead chose to use *Dynamips* directly.

Dynamips is open-source, well-documented, and actively supported by the developers of GNS3. Because of its ample documentation and because *Dynamips* hypervisors accept commands sent over a control socket, we were able to conduct several quick experiments manually creating small networks, leveraging only *Dynamips* without the GNS3 front-end. We were assisted in this process by the GNS3 debugging feature, which provides the explicit command strings sent to the *Dynamips* hypervisors for GNS3 tasks.

As with our experiments using GNS3, our initial aim was to programmatically construct a small network architecture, consisting of only three or four routers. Again, we implemented our tests in Python, and sent commands to a single *Dynamips* hypervisor over its control socket. Our tests were successful in creating small network topologies, so we again scaled our topology generation to the order of several hundred Cisco 7200 routers. In contrast with GNS3, we noticed significantly less boot-up time - on the order of five to ten minutes using pure *Dynamips*, as opposed to about an hour with our GNS3 solution. Integration with *VirtualBox* VMs with our *Dynamips* solution proved to be straightforward - our *VirtualBox* host was configured with a bridged adapter to a tunnel interface connected to our topology. Unlike GNS3, this tunnel interface is easily added and removed from a particular router vantage point through a short sequence of commands provided to the *Dynamips* hypervisors over their control socket. Lastly, the amount of human interaction necessary in this model was greatly reduced; we simply run *scamper* as a daemon listening on the VM host address on a particular port. In ERIK, we interact with this daemon by importing a Python-pure implementation of an associated *scamper* program called *sc_attach*.

Dynamips Modifications

Though a *Dynamips*-pure implementation of ERIK had eliminated or reduced many of the concerns our GNS3 model raised, we did experience two issues. First, automating vantage point changes was hindered by a bug in a *Dynamips* provided method called `vm send_con_msg()` that was not widely used or tested. This method's function is to send messages to the console of a virtualized router, and in our use case, was being used to push new base64 encoded configuration files to routers to either:

- Prepare a router for use as a vantage point. This entails pushing a new configuration file indicating a new interface for the connection to the VM monitor, assigning IP addresses, and updating routing protocol configurations, in addition to retaining all of the router to router connections this router had previously.
- Remove the router to monitor connection once all experiments had been run from that vantage point. This entails removing the routing protocol-specific configuration

details, as well as removing the interface and IP address configuration for the link to the monitor.

Initial experimentation with `vm send_con_msg()` proved that while this command would successfully transmit the base64 configuration string to the console of the indicated router, any trailing carriage return characters were not. This meant that the command would still require an operator to telnet to the router on its control socket and manually execute the command. Upon informing the *Dynamips* developers of this issue, a fix was implemented [30], allowing us to easily automate vantage point changes.

Our last experiments with *Dynamips* were tailored to designing faults within our emulated network. It is often useful to study the effects of various equipment or link failures on overall reachability, and routing table churn can cause anomalous paths to be inferred by *scamper*. Simulating a router failure is trivial with *Dynamips*; routers can easily be shut down, or a configuration file that shuts all router interfaces can be pushed via the `vm send_con_msg()` command, for example. As a design choice, however, we made the decision to study link failures rather than router failures, and to implement the ability to induce a set of link failures within ERIK. Our justification for this design decision was based on our desire to maximize the effects of these failures while running topology inference testing. In a router to router connection, the failure of a one of these endpoints will be quickly recognized by the other router due to the absence of an electrical signal over the link. However, when simulating link failures between two routers, we can increase the time it takes before this failure is discovered by placing intermediate switches between the two, and removing the link connecting the two switches. Both routers will still detect their physical connection to their associated switch, and only when a fixed period called a "dead interval" has elapsed without receiving a "hello message" from the other router will the routers detect that the link is down. Our link failure model is depicted graphically in Figures 3.2 and 3.3. A final hurdle to implementing ERIK occurred when inducing a number of link failures to occur in quick succession. During our tests, we caused a deadlock between two threads in *Dynamips* when removing links between Ethernet switches, which resulted in ERIK hanging indefinitely [31]. Changing a blocking `pthread lock` to a `trylock` ultimately resolved this issue, and was accepted as a patch by the *Dynamips* developers.

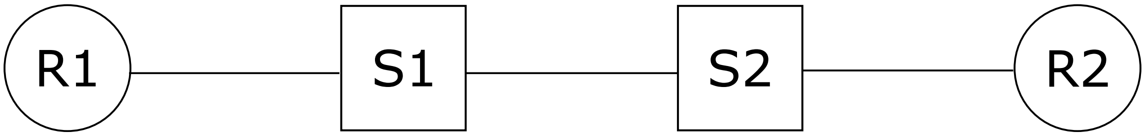


Figure 3.2: Initial state of a router to router connection selected to fail during ERIK topology inference. Each router is connected to a *Dynamips*-provided Ethernet switch, which are connected to each other by a third link. The third link connecting the two switches will fail; however, the two routers will not detect the link failure until the window in which a "hello-message" from the other router should have been received expires.

Figure 3.3: The path between the two routers after the link has been cut. The interfaces connected to the switch on each router will still be powered; each router assumes connectivity to the other until the "dead timer" expires.

Having described the early experiments and initial attempts at creating a tool for automated emulated network topology creation and inference, we now describe the current state of ERIK, and compare it to similar software.

3.2 ERIK

While ERIK is a complete package for the automated topology inference experimentation, it can be broken into two logical phases - the creation of the network to be emulated, and the actual emulation and experimentation.

3.2.1 Network Topology Generation

The topology generation phase of ERIK takes several inputs. First, the network model must be specified. Early phases of our work focused on creating "flat", or intra-AS, topologies using the Barabási-Albert and Waxman models described in Sections 2.3.1 and 2.3.2, and Erdős-Rényi random graphs as a baseline. For these network models, the number of routers desired in the topology must also be specified, as well as the α and β constants for the Waxman model. In later testing, we primarily used the tiered model described in Section 2.3.5 for generating topologies, with its associated probability parameters. Regardless of the generation model, whether link failures will be studied is a mandatory boolean parameter.

These parameters are fed into the topology creation module of ERIK. With these inputs, a *NetworkX* graph is created in the following stages:

1. The number of Tier 1, Tier 2, and customer ASs are calculated given the number of ASs and fractions specified.
2. Given n Tier 1 ASs, a random number of edges e is selected between $n-1$ and $\frac{n(n-1)}{2}$. An Erdős-Rényi random graph is created with n vertices and e edges, and if it is connected, we move to the next step. Otherwise, we repeat this process until a connected Tier 1 backbone has been generated.
3. Tier 2 ASs are connected to the Tier 1 backbone one at a time. First, the number of links a Tier 2 AS will form to Tier 1 nodes is calculated based upon the input parameters; in practice, we select a number between one and three from a Gaussian distribution. The Tier 1 nodes these connections will terminate at are selected uniformly randomly, and these edges are added to the graph.
4. When all Tier 2 ASs have been connected to their Tier 1 providers, a random sample of the Tier 2 nodes is selected to form peering links with each other based on the input parameter. These edges are added to the graph.
5. Customer ASs are added to the graph one at a time, with one or two Tier 2 provider links based on the input probability. These nodes and edges are added to the graph.

If link failures are to be studied, a final step occurs:

6. Edges in the graph are sorted by *edge betweenness centrality*, which measures the fraction of shortest paths that contain an edge. The top x of these edges are identified to be link failures during the emulation phase of ERIK, where x is a predefined parameter. Mathematically, *edge betweenness centrality* is defined in [29], [32] as:

$$b(e_i) = \sum_{s,t \in V(G)} \frac{\sigma(s,t|e_i)}{\sigma(s,t)}$$

   where $\sigma(s,t)$ is a geodesic in a graph G between vertices $s \neq t$, and $\sigma(s,t|e_i)$ is a geodesic between $s \neq t$ containing the edge e_i.

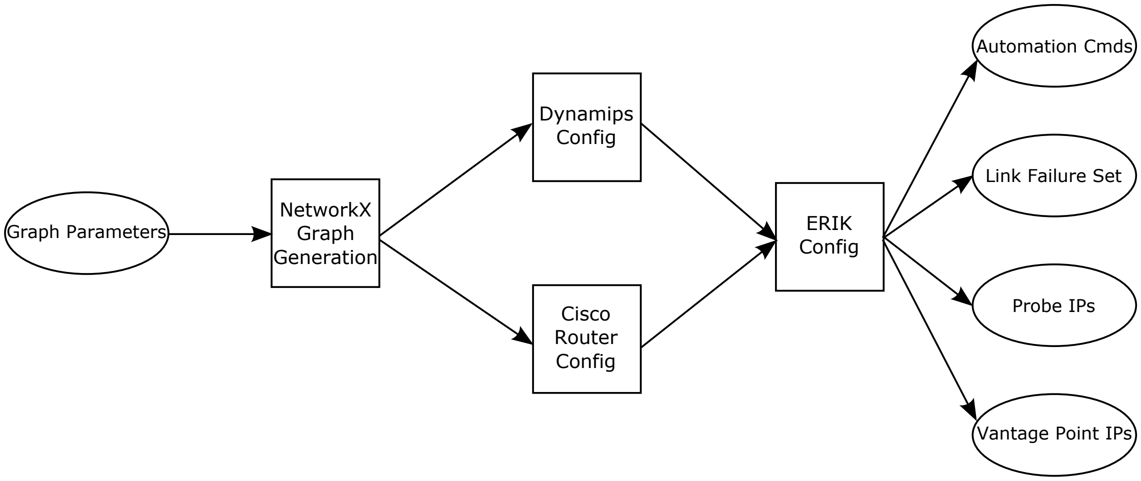


Figure 3.4: The topology generation phase of ERIK takes various graph parameters as inputs, creates a graph, and outputs a set of files. The `Automator` file lists the *Dynamips* commands necessary to run the emulation and the `VP IPs` file lists the IP addresses for the inference vantage point to take during experimentation. `Probe IPs` contains the set of IP addresses to be probed by *scamper* and `Link Failure Set` contains the names of switches that will have link failures during the emulation.

With our *NetworkX* graph created and set of link failures identified, we proceed to automating the configuration of the ASs. This is depicted graphically in Figure 3.5, and proceeds in the following manner:

1. We iterate through each edge in the graph, classifying it based on the tier to which its adjacent vertices belong, e.g., Tier 1 to Tier 2 customer-provider link, or Tier 2 to Tier 2 peering link. From these characterizations, we define route-maps that allow us to create a BGP policy model that generalizes that found on the Internet - to prefer routing to customers over peers, and to prefer routing to peers over providers. Specifically, for Tier 2 ASs, we used the route-map syntax

   ```
   ip as-path access-list 10 deny ^x_
   ip as-path access-list 10 permit .*
   ```

   for outbound interfaces on connections to Tier 1 ASs ‘x’ and Tier 2 peers. Converted to BGP routing configuration commands, this policy is stored individually for each AS.
2. We allocate IPv4 addresses to each AS according to its tier. Tier 1 backbone networks are assigned a Class A network, Tier 2 transit-provider networks are assigned a Class

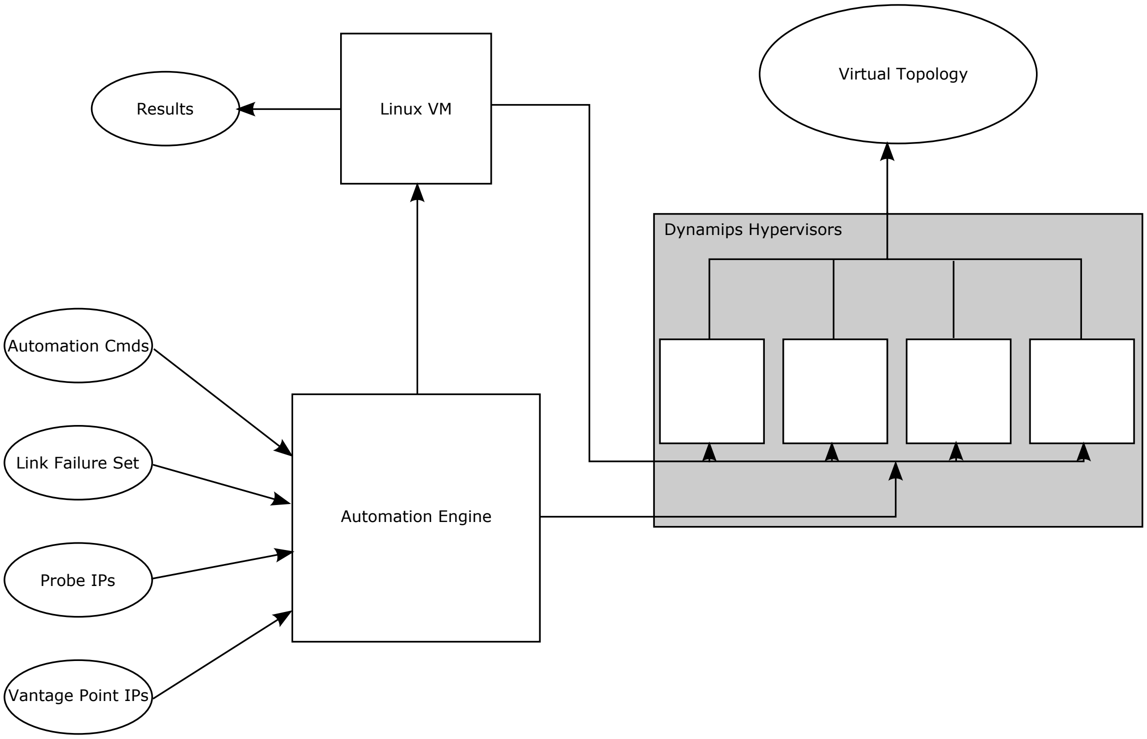
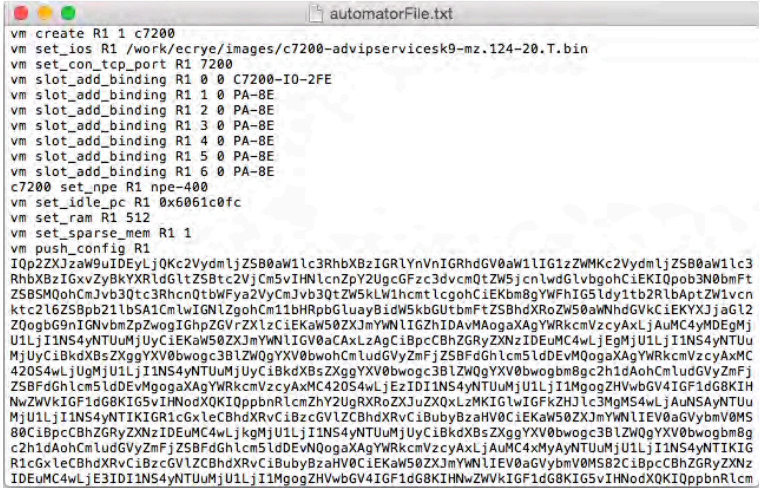


Figure 3.5: The topology emulation phase of ERIK uses the outputs from the topology generation phase as inputs. The Automation Engine reads the file `Automation Cmds` to assign routers to *Dynamips* hypervisors, and send the requisite commands to load these routers with their configuration files and create links between them. During topology inference testing, the Automation Engine sends the list of `Probe IPs` to the Linux VM to execute. During the second round of probing, the Automation Engine also induces link faults by reading the `Link Failure Set` file, and sending the appropriate *Dynamips* commands to the appropriate hypervisor. After each vantage point has undergone the three probing rounds, the Automation Engine restores the failed links, sends the next vantage point IP from file to the VM, removes the attachment to the virtualized topology at the old vantage point, and reattaches the VM to the next vantage point.

B subnetwork within one of their provider's Class A networks, and customer ASs are assigned a Class C network within a Tier 2 provider's Class B subnet. Customer-provider links are addressed out of the provider's network, and peering links are addressed out of the private Class A network. These networks and IP addresses are incorporated into the configurations started for each AS in step 1. For example, if a Tier 1 ASs assigned the `13.0.0.0/8` subnetwork, its first Tier 2 would be given the `13.1.0.0/16` Class B subnetwork. In turn, the first customer AS to be allocated IP space from that Tier 2 would be given `13.1.1.0/24`.

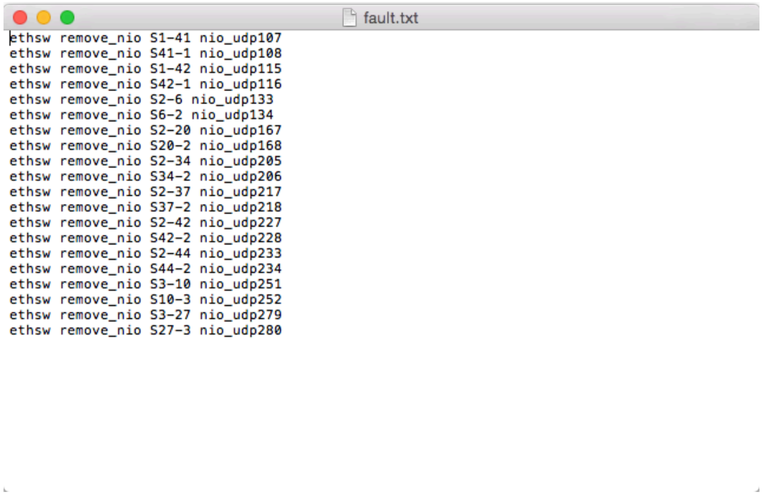


```
vm create R1 1 c7200
vm set_ios R1 /work/ecrye/images/c7200-advipservicesk9-mz.124-20.T.bin
vm set_con_tcp_port R1 7200
vm slot_add_binding R1 0 0 C7200-IO-2FE
vm slot_add_binding R1 1 0 PA-8E
vm slot_add_binding R1 2 0 PA-8E
vm slot_add_binding R1 3 0 PA-8E
vm slot_add_binding R1 4 0 PA-8E
vm slot_add_binding R1 5 0 PA-8E
vm slot_add_binding R1 6 0 PA-8E
c7200 set_npe R1 npe-400
vm set_idle_pc R1 0x6061c0fc
vm set_ram R1 512
vm set_sparse_mem R1 1
vm push_config R1
```

```
IQp2ZXJzaW9uIDEyLjQKc2VydmljZSB0aW1lc3RhbXBzIGRlYnVnIGRhdGV0aW1lIG1zZWMKc2VydmljZSB0aW1lc3
RhbXBzIGxvZyBkYXRldGltZSBtc2VjCm5vIHNlcnZpY2UgcGFzc3dvcmQtZW5jcnlwdGlvbgohCiEKIQpob3N0bmFt
ZSBSMQohCmJvb3Qtc3RhcnQtbWFya2VyCmJvb3QtZW5kLW1hcmtlcgohCiEKbm8gYWFhIG5ldy1tb2RlbApoZW1vcn
ktc2l6ZSBpb21lbSA1CmlwIGNlZgohCm11bHRpbGluayBidW5kbGUtbmFtZSBhdXRoZW50aWNhdGVkCiEKYXJjaGl2
ZQogbG9nIGNvbmZpZwogIGhpZGVrZXlzCiEKaW50ZXJmYWNlIGZhIDAvMAogaXAgYWRkcmVzcyAxLjAuMC4yMDEgMj
U1LjI1NS4yNTUuMjUyCiEKaW50ZXJmYWNlIGV0aCAxLzAgCiBpcCBhZGRyZXNzIDEuMC4wLjEgMjU1LjI1NS4yNTUu
MjUyCiBkdXBsZXggYXV0bwogc3BlZWQgYXV0bwohCmludGVyZmFjZSBFdGhlcm5ldDEvMQogaXAgYWRkcmVzcyAxMC
42OS4wLjUgMjU1LjI1NS4yNTUuMjUyCiBkdXBsZXggYXV0bwogc3BlZWQgYXV0bwogbm8gc2h1dAohCmludGVyZmFj
ZSBFdGhlcm5ldDEvMgogaXAgYWRkcmVzcyAxMC42OS4wLjEzIDI1NS4yNTUuMjU1LjI1MgogZHVwbGV4IGF1dG8KIH
NwZWVkIGF1dG8KIG5vIHNodXQKIQppbnRlcmZhY2UgRXRoZXJuZXQxLzMKIGlwIGFkZHJlc3MgMS4wLjAuNSAyNTUu
MjU1LjI1NS4yNTIKIGR1cGxleCBhdXRvCiBzcGVlZCBhdXRvCiBubyBzaHV0CiEKaW50ZXJmYWNlIEV0aGVybmV0MS
80CiBpcCBhZGRyZXNzIDEuMC4wLjkgMjU1LjI1NS4yNTUuMjUyCiBkdXBsZXggYXV0bwogc3BlZWQgYXV0bwogbm8g
c2h1dAohCmludGVyZmFjZSBFdGhlcm5ldDEvNQogaXAgYWRkcmVzcyAxLjAuMC4xMyAyNTUuMjU1LjI1NS4yNTIKIG
R1cGxleCBhdXRvCiBzcGVlZCBhdXRvCiBubyBzaHV0CiEKaW50ZXJmYWNlIEV0aGVybmV0MS82CiBpcCBhZGRyZXNz
IDEuMC4wLjE3IDI1NS4yNTUuMjU1LjI1MgogZHVwbGV4IGF1dG8KIHNwZWVkIGF1dG8KIG5vIHNodXQKIQppbnRlcm
```

Figure 3.6: A brief section of an `Automator` file used to emulate a 300-router topology. At the bottom, the *Dynamips* command to push a base64-encoded Cisco router configuration file to a device is shown.

3. Each AS is connected to one Ethernet switch; the interface connecting the AS to this switch is assigned an IP address out of that ASs network. These IP addresses are written to a `Probe IP` file that will be used as the destinations for our *scamper* probes in the topology inference phase of ERIK. For convenience, all of the interfaces to the switches are assigned the `.1` address from the ASs network.
4. A list of vantage point IP addresses is generated using the `.2` address from the ASs network. During the emulation phase, the monitor will infer topology from each AS; its connection to the emulated topology will be through each ASs single switch.
5. The links identified to fail during the emulation phase are created with two intermediate switches, as in Figure 3.2. The names of the connections between intermediate switches are saved to a `Link Failure Set` file; Figure 3.7 shows an example in which ten links have been selected for failure.
6. Each ASs configuration is converted into a Cisco router configuration file through simple string manipulation. These files are converted to base64 encoding for use in the following step.
7. Finally, a file consisting of the *Dynamips* hypervisor commands necessary to start the topology and run inference experimentation is generated. This consists of a bootup phase, in which each AS is allocated the appropriate virtual hardware, memory and configuration file, and an emulation phase that changes the virtual connec-



```
ethsw remove_nio S1-41 nio_udp107
ethsw remove_nio S41-1 nio_udp108
ethsw remove_nio S1-42 nio_udp115
ethsw remove_nio S42-1 nio_udp116
ethsw remove_nio S2-6 nio_udp133
ethsw remove_nio S6-2 nio_udp134
ethsw remove_nio S2-20 nio_udp167
ethsw remove_nio S20-2 nio_udp168
ethsw remove_nio S2-34 nio_udp205
ethsw remove_nio S34-2 nio_udp206
ethsw remove_nio S2-37 nio_udp217
ethsw remove_nio S37-2 nio_udp218
ethsw remove_nio S2-42 nio_udp227
ethsw remove_nio S42-2 nio_udp228
ethsw remove_nio S2-44 nio_udp233
ethsw remove_nio S44-2 nio_udp234
ethsw remove_nio S3-10 nio_udp251
ethsw remove_nio S10-3 nio_udp252
ethsw remove_nio S3-27 nio_udp279
ethsw remove_nio S27-3 nio_udp280
```

Figure 3.7: A full `Link Failure Set` file for a 300-router topology. In this file, the 20 entries correspond to both directions of ten links that were selected to fail based on their edge betweenness centrality. *Dyamips* uses a "Network Input/Output" (NIO) descriptor for each side of a link.

tion from the monitor to each AS in the topology. These commands are written to the `Automator` file; Figure 3.6 shows a small section of an `Automator` file used to generate a 300-router topology.

The general outline of work flow is given in Figure 3.4.

3.2.2 Emulation and Topology Inference

Following the generation of the outputs from the topology generation phase of ERIK, we begin the topology inference experimentation with *scamper*. Prior to the emulation starting, a Linux VM is booted with *VirtualBox*; this VM will act as the monitor to probe destination ASs within our virtual topology. We chose a minimal distribution of Ubuntu in order to reduce the amount of Central Processing Unit (CPU) and memory utilized by our monitor. A *scamper* daemon and a persistent *netcat* listener are started on predefined ports, with the *netcat* daemon acting as a mechanism to pipe interface and route configuration changes to the VM during testing. Further, a BIRD daemon is started that will inject approximately 50,000 BGP routes into our virtualized topology for added realism and to induce additional router RIB/FIB churn during link failures. Finally, this VM is allocated two bridged connections to `tap` interfaces on the host.

The inputs to the emulation and inference module of ERIK consist of the outputs from the generation step. Prior to any emulation, a number of *Dynamips* hypervisors must be initialized. Through a trial and error process, we found that a set of fifteen hypervisors is capable of emulating the 300 router topologies of interest to this thesis. Hypervisor allocation depends both on the number of routers and number of links between them, however; this general guideline may not be sufficient in all cases. Regardless, the emulated routers are spread evenly across the set of *Dynamips* hypervisors.

ERIK begins the emulation by reading the `Automator` file, which contains the *Dynamips* commands necessary to initialize each router, allocate its memory, and create the links between them. Further, the interfaces between the emulated topology and host are created so that the Linux VM can conduct inference probing and inject BGP routes via BIRD. The router bootup phase ends when a termination token is read in the `Automator` file, at which point a predefined time period is allowed to elapse in order to allow for configuration loading and topological convergence.

Following the bootup and convergence time, topology inference testing begins. With the monitor VM connected to the first AS, we probe the topology in three phases. Regardless of the phase, ERIK connects to the *scamper* daemon running on the Linux VM, and passes it the list of IP addresses to probe contained in the `Probe IPs` file. We describe each of the three probing rounds here:

1. The first round of probing occurs when the topology has converged. After the call to the *scamper* daemon to probe returns, the first probing round has terminated. The results are written in *warts* binary format for later analysis.
2. Following the first probing round, the second probing round begins. As with the first round, it is initiated by ERIK connecting to the *scamper* daemon running on the monitor VM, and passing it the IPs addresses to probe. During the probing, however, a separate ERIK thread interfaces with the *Dynamips* hypervisors and removes the links between routers identified during the topology generation phase; these are read from the `Link Failure Set` file. When the call to *scamper* running on the VM returns, the second round of probing is concluded; the VM writes the results to disk.
3. The final round of probing begins after an experimentally-defined time period has elapsed in order to allow topological reconvergence following the link failures in

second probing round. Generally, we find fifteen minutes to be sufficient. After this reconvergence period, the third probing round begins to capture the topology of the network following the link failures. Finally, the monitor VM writes these results in *warts* format.

After the final round of probing is complete, the links that have failed are reconnected, and the routers are allotted a period of time to rediscover these links, and update their routing tables. This resets the virtualized, emulated environment so that network inference probing can be conducted at another vantage point, beginning with the first round described above.

These three rounds of probing provide a snapshot of the inferred topology before, during, and after failure events as inferred from the monitor at its vantage point within the topology. We capture these three snapshots for two reasons. First, the inferred graphs provide insight into commonalities and differences between these three states; in essence, how do various failures effect the structure of the inferred network, and what remains the same? Second, because we generated topological ground truth when we created the topology, we can compare the three inferred graphs against the actual physical structure of the network - a comparison we cannot make for inferred Internet topologies.

Finally, ERIK performs these three probing rounds exhaustively - once completed at the first vantage point, ERIK reads the appropriate commands from the `Automator` file and sends them to the *Dynamips* hypervisors to move the monitor to the next vantage point. In addition, ERIK passes the requisite interface configuration and default gateway changes to the monitor from the `VP IPs` file. After these adjustments are complete, probing begins anew.

After probing has been conducted at each vantage point, ERIK terminates, and the result data from the inferred topologies can be analyzed, as we will see in Chapter 5. Because ground truth is known, router alias resolution is trivial; we produce *Gephi* graphs for each inferred topology for visualization purposes [33].

Finally, we present pseudocode of two subcomponents of ERIK, namely the topology generation and emulation automation phases. In Algorithm 1, we describe our tiered topology generation model, given input parameters n_1, n_2 and n_3 – the number of Tier 1, Tier

Algorithm 1 Tiered Topology Generation Pseudocode

```
do
    Choose random integer $m$ such that $n_1 - 1 \leq m \leq \frac{n_1(n_1-1)}{2}$
    $G$ = Erdős-Rényi random($n_1, m$)
while $G$ disconnected
for $node$ in Tier 2 nodes do
    while $deg(node) < tier2degree$ do
        Randomly select node $v_1$ in Tier 1 nodes
        Add $(node, v_1)$ to $E(G)$
for $node$ in Tier 2 nodes do
    while Number of peers $< tier2peers$ do
        Randomly select node $v_2$ in Tier 2 nodes
        Add $(node, v_2)$ to $E(G)$
for $node$ in customer nodes do
    while $deg(node) < custdegree$ do
        Randomly select node $v_2$ in Tier 2 nodes
        Add $(node, v_2)$ to $E(G)$
return $G$
```

2 and customer ASs desired, $tier2degree$, the Tier 2 ASs to Tier 1 ASs average degree, $custdegree$, the average customer ASs to Tier 2 ASs degree, and $tier2peers$, the Tier 2 ASs peering degree. In Algorithm 2, we detail the high-level idea of the ERIK automation engine.

Algorithm 2 Network Emulation Pseudocode

```
//Topology Start-up Phase
do
    Read next router's Dynamips configuration
    Send router configuration commands to appropriate hypervisor
    for Link terminating at router do
        Send link configuration commands to hypervisor
while !End start-up
Sleep while topology boots and converges
//Topology Inference Phase
for router in topology do
    Attach VM monitor via tap interface
    Initial probe of topology; write results
    Begin second probe, sending commands to fail links during probing; write results
    Sleep while topology reconverges with failed links
    Probe topology; write results
    Restore failed links, sleep while topology reconverges
```

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Internet Graph Reduction Methodology

In this section, we describe our methodology for extending previous work by Krishnamurthy, Faloutsos, *et al.* in [27]. Recognizing that this work relied upon one data set, namely the RouteViews BGP RIB tables from the 2001 to 1998 timeframe, we first endeavored to determine whether the graph reduction methods described in 2.3.6 were still valid over a longer, more current timeframe, and whether using a different data set might yield the same conclusions. Further, we develop two new graph reduction algorithms designed to replicate Internet graph properties at a smaller scale.

4.1 Validation and Extension of Prior Graph Reduction Work

In order to validate that our algorithms for the graph reduction methods described in Section 2.3.6 were correct, we first replicated the experiments in [27]. We start with an initial, largest Internet instance from 7 May 2001, and reduce this graph to a graph with the same number of nodes as the inferred Internet instance on 24 January 1998, using each of the reduction methods described in Section 2.3.6. These dates were specifically chosen to match the work of Krishnamurthy, Faloutsos, *et al.* in [27]. Our data, as in prior work, was obtained from the RouteViews archives. Following the replication of previous experiments, our work extends the body of knowledge in two ways.

First, we endeavored to determine whether our findings, and the findings of [27], extend to more current Internet instances. Toward this goal, we conducted the same experiments beginning with an initial, largest Internet instance from 1 December 2014 obtained from the RouteViews archive. This graph was reduced to a graph composed of the same number of vertices as the 24 January 1998 Internet instance. Again, we examine the results of reducing this graph using all methods described in Section 2.3.6.

Secondly, we expand the data sets considered by reproducing these experiments leveraging CAIDA-compiled AS-level topologies. Because the RouteViews and CAIDA AS-level

Internet graphs are obtained in distinct ways and imperfectly capture the topology of the Internet, we believe that a reduction algorithm should reduce Internet graphs from both data sources reasonably well in order to be considered an effective method for creating Internet-like graphs.

To reduce terminology confusion, we refer to each of the different data source and time period combinations as a distinct *source-period*. In particular, the four source-periods considered are the RouteViews data from 2001-1998 and 2014-1998, and the CAIDA data from 2014-1998 and 2001-1998. In all four source-periods, we consider the graph reduction methods' effectiveness in the following categories:

1. Average degree of the reduced graphs compared to Internet's reduced graph over time.
2. Spectral analysis of the 100 largest eigenvalues of the normalized adjacency matrix of the most reduced graphs per reduction methods. These spectra are compared to the Internet spectra from January 1998.
3. *Hop-plot* or plot of cumulative vertex-pairs against the number of hops apart these pairs are along a geodesic in the most reduced graph. This is compared to the January 1998 Internet *hop-plot*.
4. A sorted degree-distribution of the most reduced graphs versus the degree distribution of the January 1998 instance.

Our methodology for evaluating these metrics follows:

1. For each source-period, we extracted a set of intermediate Internet instances between the initial, most recent Internet instance, and the end point for the reduction in January 1998. These intermediate instances formed target vertex set cardinalities to save partially reduced graphs at during the reduction process by the methods described in Section 2.3.6. At each intermediate vertex count, we calculated the number of edges contained in our graph reduced by each method to obtain an average degree. This was then plotted against the Internet's average degree over time. For example, in the Routeviews 2001-1998 source-period, we chose intermediate Internet instances equally spaced in time between May 2001 and December 1998. When our reduction methods had reduced the May 2001 graph to the graph order, or number of vertices

in the vertex set, of these intermediate instances, we calculated the average degree of our reduction-method graph. This average degree is then compared to the actual Internet degree at that point in time.

2. In each source-period, we compute the 100 largest eigenvalues of the normalized adjacency matrix for the reduction endpoint in January 1998, and plot these in decreasing order. For each reduction algorithm, we compute the eigenvalues of the graph's normalized adjacency matrix at the reduction endpoint - the graph of the order of the January 1998 Internet instance - and plot the sorted eigenvalues against the Internet's.
3. For *hop-plot*, we calculate the cumulative fraction of vertex pairs that are within *k* hops along a geodesic in the January 1998 Internet graph, where *k* ranges between 1 and the diameter of the Internet. We then compute the *hop-plot* curve for each of the reduction methods' most reduced graphs, i.e., those of order equivalent to the January 1998 Internet graph. We plot these to compare similarity of shortest paths between graphs.
4. Again, for degree distribution, we compare only the most-reduced graphs from each reduction method against the values of the Internet graph at its chronologically earliest instance considered.

For each reduction method, we compute and plot the average of fifty trials using distinct random seeds to reduce the statistical influence of outliers.

4.2 Novel Graph Reduction Algorithms

Motivated by previous work that found *k*-cores of AS-level Internet graphs to be self-similar [34], we undertook development of new graph reduction algorithms that exploit this self-similarity. The standard definition of a *k*-core of a graph is:

Definition Given a graph G with vertex set $V(G)$, the *k*-core of G is the maximal connected subgraph H of G such that

$$deg(v) \geq k, \; \forall v \in V(H).$$

In other words, a graph *k*-core is a connected subgraph in which all nodes have degree at least *k*; if multiple such subgraphs exist, the *k*-core is the largest such subgraph by number of nodes. Given a graph G with n nodes, the complexity of finding the k-core of the graph is $O(n)$, as it can be obtained by repeatedly deleting nodes of degree less than k.

Unlike the sampling methods described in Section 2.3.6, our algorithms require a second parameter in addition to the number of vertices to reduce a graph to; both require a target edge number as well. This additional input parameter is both a benefit and a disadvantage; in specifying a target edge number to attain, we ensure that the average degree of our reduced graph matches that of the January 1998 Internet graph. However, the edge parameter is disadvantageous in that we now require a second parameter at all; when reducing graphs for the purpose of emulation, we are at best providing an educated guess for the number of edges. This could be accomplished by fitting a curve to the average degree of the Internet line over time and extrapolating the data point at the desired number of ASs. Indeed, even the CAIDA and RouteViews Internet graphs provide an incomplete picture of the AS-level topology as well, as they suffer from the topology inference limitations discussed in Chapter 2.

We describe our methods, termed k-core decomposition/DRVE/DRE (KDD) and k-core decomposition/k-deletion/DRE (KKD), here.

4.2.1 KDD

The force behind the development of KDD stemmed from a desire to capture the most topologically important, highest-degree ASs in the Internet graph, while removing less significant nodes based on degree. While degree is an imperfect indicator of graph structure and node importance, the highest-degree nodes most likely take part in the largest clusters within the graph, which we considered a desirable graph property to maintain. Further, our intuition was that relatively high-degree vertices were more likely to have existed in earlier Internet graphs than lower-degree nodes, against which we ultimately compare our reduced graphs. Motivated by this intuition, we examined k-cores of the Internet graphs.

Simply sampling the initial Internet graph by finding k-cores for increasing values of k, however, resulted in several problems. First, a naïve k-core reduction is not fine-grained in that we cannot specify an exact number of nodes to which to sample a graph – given a number of nodes n desired in the reduced graph, it is unlikely that there exists a k for which the number of nodes in the k-core is equal to n. For this reason, we cease reducing the original graph by retaining k-cores at the smallest value of k for which the number of nodes in the k-core is greater than the number of nodes we desire. At this juncture, we may have to remove a non-trivial number of nodes in order to reach the desired node count; for this purpose, we leverage DRVE, which selects a node from the node set with a uniformly random probability, and removes an edge incident to that node. Alternative methods could be considered for removal of the excess nodes before reaching the number of nodes required in the reduced graph; in Section 4.2.2 we explore another.

In practice, our sampled Internet graphs had far too many edges at this point, though the correct number of nodes required for our reduction. Ending our reduction without addressing the excess edges would result in a graph with an average degree several times that which we were attempting to reach. Therefore, it was necessary to apply one further reduction to only the edges in the graph. Because we achieve the desired node count in the second step by DRVE, it was necessary to sample edges in a manner that did not result in disconnections of the graph and reduce the node count. To accomplish this task, we remove edges by selecting an edge uniformly at random, and removing it only if the graph induced by the edge removal remains connected, and repeating until the desired edge count has been obtained.

In summary, KDD is composed of the following steps:

1. The first stage consists of computing k-cores of the initial graph for $k = 1, 2, 3, \ldots$; we stop when the $(k+1)^{st}$-core contains fewer than the target number of nodes for the reduced graph. Equivalently, we compute k such that the k-core of the initial graph has more nodes than we want in our reduced graph, and the $(k+1)^{st}$-core contains less nodes than we require in our reduced graph. We retain the smallest k-core in terms of node count (largest k value) with more nodes than we require for our reduction,
2. Next, we apply DRVE until we reach the number of nodes required in the reduced graph.
3. Modified DRE; remove a randomly selected edge if it does not result in a graph disconnection. Repeat until the required number of edges in the reduced graph has been achieved.

Pseudocode for the KDD reduction method is given in Algorithm 3.

Algorithm 3 KDD graph reduction

```
Given an initial graph $G$ and reduction endpoint graph $T$
//$k$-core reduction of $G$
$k \leftarrow 1$
$H \leftarrow k$-core($G$)
while $|V((k+1)\text{-core}(G))| > |V(T)|$ do
    $k \leftarrow k+1$
    $H \leftarrow k$-core($G$)
//DRVE phase to reduce node count
while $|V(H)| > |V(T)|$ do
    Apply DRVE
//DRE phase to reduce edge count
while $|E(H)| > |E(T)|$ do
    Apply modified DRE
return $H$
```

Complexity for KDD is $O(|V| + |V||E| + |E|)$. Finding k-cores is linear in the number of nodes, as noted previously, while DRVE scales as $O(|V||E|)$ because it involves first selecting a random vertex, then choosing an edge incident to that vertex. DRE scales with $O(|E|)$, as it entails removing a uniformly random edge from the graph.

4.2.2 KKD

In Section 4.2.1, we chose to utilize DRVE following a reduction of an original graph instance by examining k-cores with increasing values of k. When we obtain the smallest k-core of the initial graph that contains more nodes than the reduction endpoint graph, we remove excess nodes via DRVE. In Section 4.2.1, we observe that this is by no means the only method for achieving a reduction of nodes, and in this section, we present another mechanism.

In KKD, we begin reduction of the initial graph instance G as before by obtaining progressively smaller k-cores by increasing the value of k. This phase terminates when $(k+1)$ would result in a core with less nodes than we require in our reduced graph.

At this point, KKD diverges from KDD. Rather than leveraging DRVE to reduce the node count of the graph, KKD deletes nodes in the graph with degree k until the number of nodes in the graph matches that of the reduced graph. The intuition for this approach was to attempt to retain the highest-degree nodes in the graph while removing those of lower degree, to examine an alternative method for node count reduction to DRVE used in KDD.

Finally, KKD suffers from the same excess number of edges problem we observed with KDD, and we resolve it in the same manner as before – by removing random edges only if their removal does not result in a disconnection of the graph. When the edge count is reached, a reduction by KKD is complete.

The following is a brief summary of the KKD reduction algorithm, and the pseudocode for KKD is given in Algorithm 4.

1. The first stage consists of computing k-cores of the initial graph for $k = 1, 2, 3, \ldots$; we stop when the $(k+1)^{st}$-core contains fewer than the target number of nodes for the reduced graph. Equivalently, we compute k such that the k-core of the initial graph has more nodes than we want in our reduced graph, and the $(k+1)^{st}$-core contains less nodes than we require in our reduced graph. We retain the smallest k-core in terms of node count (largest k value). This step is identical for KDD and KKD.
2. Next, we proceed by randomly removing nodes whose degree is exactly k from the graph obtained from the prior k-core selection process. Nodes are removed until the target node number is reached.
3. Finally, the last stage consists of removing the excess edges to reach the target edge number. In order to preserve the required number of nodes we reached in the second stage, only edges whose removal does not disconnect the graph are removed. Edge removal is repeated until the desired edge count is reached. This stage is performed in KDD as well.

As stated earlier, the complexity for finding k-cores of a graph is linear in the number of nodes. The deletion of nodes with $deg = k$ is linear in the number of nodes as well, and the modifiedDRE phase scales with the number of edges in the graph. Therefore, the complexity of KKD is $O(2|V| + |E|)$.

Algorithm 4 KKD graph reduction

```
Given an initial graph $G$ and reduction endpoint graph $T$
//$k$-core reduction of $G$
$k \leftarrow 1$
$H \leftarrow k$-core($G$)
**while** $|V((k+1)\text{-core}(G))| > |V(T)|$ **do**
    $k \leftarrow k+1$
    $H \leftarrow k$-core($G$)
//$deg = k$ deletion phase to reduce node count
**while** $|V(H)| > |V(T)|$ **do**
    Randomly select $n \in V(H)$, where $deg_H(n) = k$
    Remove $n$ and its incident edges from $H$
//DRE phase to reduce edge count
**while** $|E(H)| > |E(T)|$ **do**
    Apply modified DRE
**return** $H$
```

In the next chapter, we analyze our topology inference results obtained by ERIK, our Internet graph reduction survey, and join the two by emulating a reduced Internet graph using ERIK in the next chapter.

CHAPTER 5:
Results and Analysis

In this chapter we first describe some results obtained from running the state-of-the-art Internet topology mapping software tool, *scamper*, on emulated topologies using ERIK. We then detail our AS-level Internet graph reduction results, including the performance of the novel techniques introduced in Section 4.2. Finally, we combine these these two objectives in an analysis of topology inference using ERIK on a graph reduced by the DHYB-0.7 reduction method introduced in Section 2.3.6.

5.1 Analysis of ERIK Topologies

Although ERIK is capable of and was designed to generate, emulate, and exhaustively probe topologies, we focus here on a single network topology. We do this in order to provide insight into the types of results can be obtained from ERIK, and the analyses we can undertake using those results. In this section, we specifically look at the fraction of network discovered via topology inference testing, and examine particular ASs that exhibit pathologies during inference testing to determine their root causes.

Our first detailed analysis of a network generated and emulated by ERIK is a 300-AS topology, in which each AS is modeled by a single Cisco 7200 series router. This topology is composed of 15 Tier 1, 45 Tier 2, and 240 customer ASs, connected by 676 edges. Following initialization and loading of the emulated routers, inference probing begins using the methodology described in Chapter 3. In all three rounds of probing, we probe the emulated network exhaustively from a single vantage point at a time, using *scamper* to probe to an IP address associated with every AS.

In the first round of probing at each vantage point, every AS is discovered, though the edges in the inferred topologies vary due to BGP policy. For example, a peering link connecting two Tier 2 ASs is not discovered in the inference using a Tier 1 AS as the vantage point. Discovery of all ASs in the first probing round is by design, and implemented in our BGP policy rules described in Section 3.2.1 together with our tiered model in Section 2.3.5.

In the second round of probing from each vantage point, in which 10 links between ASs fail, our inferences exhibit a wide range of ASs that go undiscovered, from 3 to 272. For example, using Autonomous System Number (ASN) 11 as our vantage point in the link failures round, our inference probing did not discover 10 Tier 2 ASs and 118 customer ASs, for a total of 128 ASs missed. This is in marked contrast to the first inference probing round, in which using ASN 11 as the vantage point discovered all 300 ASs. In contrast, ASN 17, a Tier 2 AS, discovers all ASs in the first probing round, and all but three of the ASs in the topology during the second round of probing, a difference of 125.

Figures 5.1 and 5.2 display Cumulative Distribution Functions (CDFs) of the fraction of ASs that do not return an ICMP response when probed by *scamper*. In turn, when the *scamper* output files are parsed into IP-layer paths, these ASs are missing from the inferred graph. Figure 5.1 graphically depicts the churn caused by the link failures – 50% of the nodes in the topology do not discover approximately 20% of the destinations during the failures round compared with the original probing round. In Figure 5.2, the CDF is separated by tier of the vantage point AS used for the topology probing. The long tail in both plots indicates that a small fraction of inferences from particular vantage points miss a large portion of the topology. For example, using ASN 22 during our testing leads to 272 missed ASs, over 90% of the ASs in the topology.

In summary, we make several observations about the second round of probing, during which link failures occurred:

- Every ASs, when used as a vantage point, does not discover at least some of the target ASs.
- When a Tier 1 AS is used as a vantage point, it tends to miss much less of the topology than Tier 2 or customer ASs. This is likely due to a greater diversity of paths available to Tier 1 ASs than to the other tiers.
- A small fraction of Tier 1, Tier 2, and customer ASs miss nearly all of the topology when used as a vantage point. This will be explored in more depth in Section 5.1.1.

In the final round of probing, after a period of time has expired to allow the routers within the topology to receive updated BGP routes to refresh their RIBs, we see far less variation in the number of ASs missed in the *scamper* probing. With the exception of three ASNs,

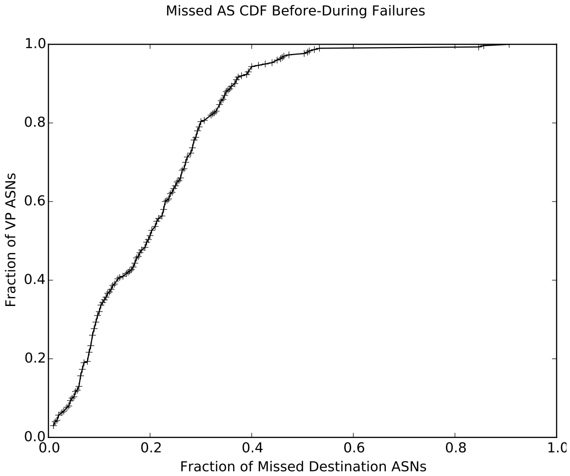


Figure 5.1: CDF of ASNs by fraction of missed ASNs during second probing round with failures

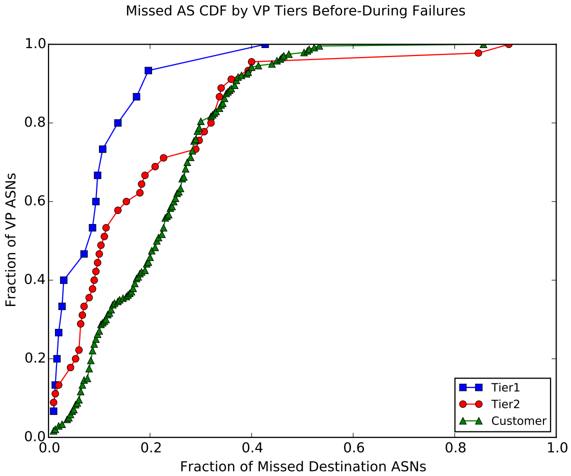


Figure 5.2: CDF of ASNs by fraction of missed ASNs during second probing round with failures, separated by vantage point AS tier

using nearly every other AS as a vantage point finds all but three ASs during probing. Our previous example that missed 128 ASs in the second round, ASN 11, misses only three ASs in this round. Because we wait a sufficient amount of time to allow the topology to reconverge following link failures, these 297 AS vantage points that miss three ASs in the final probing round do so because BGP policy prevents traffic from that vantage point to be routed to the destination; we term this effect a *policy disconnection* and discuss further in Section 5.1.2.

Figures 5.3 and 5.4 display the results of topology inference after the links have failed and the topology has been given an allotted period of time to reconverge its routing tables. Figure 5.3 shows 99% of the ASs used as vantage points discover all but 1% of the topology. The long tail indicates the other 1% of monitor ASNs missing over 90% of the ASs in the topology. Figure 5.4 displays the CDF separated by tier. In Figure 5.4, we observe that all Tier 1 nodes discover nearly all of the topology; every one of the 15 Tier 1 nodes finds 297 ASs during the third probing round. The two ASs that miss the majority of the topology come from the Tier 2 nodes, with a customer node failing to infer most of the topology as well.

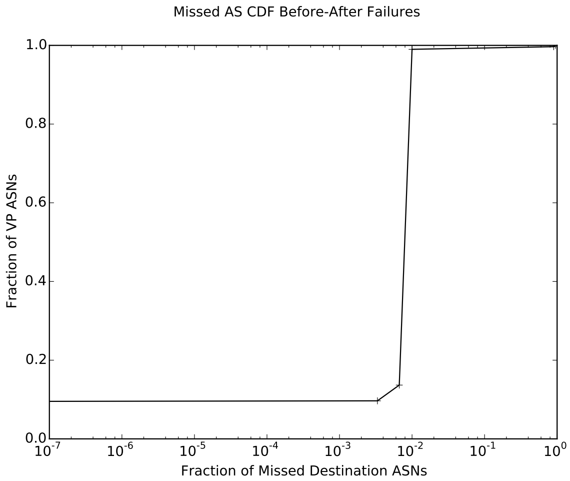


Figure 5.3: CDF of vantage point ASNs by fraction of missed ASNs during the final probing round with failures

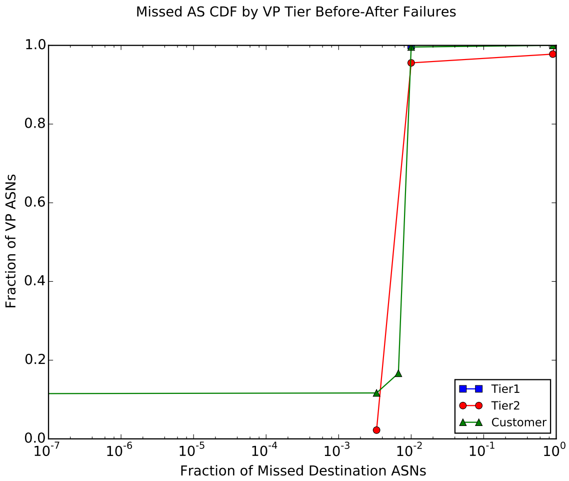


Figure 5.4: CDF of vantage point ASNs by fraction of missed ASNs during the final probing round, separated by vantage point AS tier

Next, we more closely examine ASN 11 as an exemplar of the reason for the significant change in number of ASs discovered between the first, second, and final inference rounds.

5.1.1 ASN 11 — A Closer Look

As noted previously, ASN 11 is a Tier 1 node within the topology, with a degree of 17 in the network. Further, this AS is central in the topology – seven of the ten links connecting ASs with the highest edge betweenness centrality (our criteria for selecting link failures) are incident to ASN 11. All vantage points, including ASN 11, discover every AS during the initial inference round, prior to link failures. In the second round, however, seven of ASN 11's 17 links fail. This induces a significant amount of routing table churn that must be resolved before ASN 11 can continue to route traffic to many of the destination IP addresses probed by *scamper*. Further, because ASN 11 is a Tier 1 AS in our topology, it must forward BGP route updates to the 10 ASs that remain connected following its discovery of the failed links. ASN 11 does not detect these link failures simultaneously, which increases the time required to propagate route updates throughout the network and delays routing convergence. During this period, probes to 128 ASs fail to reach their destinations due to routing table churn.

In the final topology inference round, ASN 11 discovers all but three of the ASs in the topology, a change of 125 ASs from the second probing round during which links failed. The 10 remaining links provide a route to all but three of the remaining ASs in the topology. The three ASs that remain missing in the final trace from ASN 11 are discussed further in the next section.

5.1.2 Policy Disconnections

An overwhelming majority of ASs in our topologies of §5.1 miss three ASs during the final inference probing; furthermore, these are the *same* three ASs missed from each vantage point. This is due to the effects of BGP policy, which is the focus of this section – link failures causing correctly implemented BGP policy rules used to enforce economic relationships to inadvertently disconnect ASs from the wider network.

ASNs 22 and 25, Tier 2 ASs, are both *single-homed* to different Tier 1 backbone ASs. In addition, ASNs 22 and 25 peer with each other. As part of the emulated scenario, the link from ASN 22 to its Tier 1 provider fails during the second round of inference probing.

Recall that an ASs advertises its customer routes over a peer link, but not any provider routes – this prevents the peering link from becoming free transit (to the upstream Internet) for the remote peer. Similarly, an ASs will not advertise routes learned from a peering link to its own providers. These BGP peering behaviors represent standard policy-based routing that is common on the Internet.

After the link failure, ASN 22 still has connectivity to ASN 25 over the peering link. As ASN 25 is connected its Tier 1 provider, the network graph remains connected. However, the graph is *policy disconnected*. No traffic from the Tier 1 backbone can reach ASNs 22 or 89, as depicted in Figure 5.5. Similarly, no single-homed customers of ASN 22 are reachable from any point in the network other than ASN 25.

The result of this situation is three ASs that are in effect disconnected from the remainder of the topology, despite the topology remaining connected when viewed as an undirected graph.

A graphical depiction of the topology as inferred in the third round of probing (after the topology has converged) at ASN 89, which has only ASN 22 as a Tier 2 provider, is shown

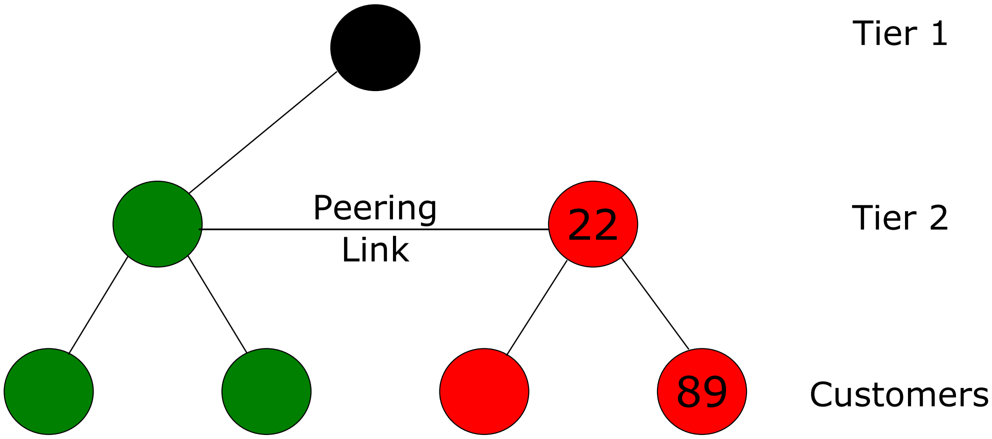


Figure 5.5: Routes to ASN 22 are not advertised to the wider network from its Tier 2 peer; therefore, it and its customers are not discovered when probed from ASs such as the Tier 1 at the top of the figure. ASs discoverable from this Tier 1 vantage point are shown in green, while those unable to be probed are colored red.

in Figure 5.6. ASN 22 is shown as the large node connected to ASN 89, while the other large red node represents ASN 22's Tier 2 peer, ASN 25. The only ASs discovered in the final inference from ASN 89 are customers of one of these two Tier 2 peers.

5.2 Internet Graph Reduction Results

In this section, we examine the results of four distinct graph reductions:

- RouteViews BGP RIB topologies reduced from 2001 instances to 1998-sized graphs (Section 5.2.1).
- RouteViews BGP RIB topologies reduced from 2014 instances to 1998-sized graphs (Section 5.2.2).
- CAIDA BGP topologies reduced from 2001 instances to 1998-sized graphs (Section 5.2.3).
- CAIDA BGP topologies reduced from 2014 instances to 1998-sized graphs (Section 5.2.4).

In comparisons of reduction methods to the Internet in spectral analysis, *hop-plot*, and degree distribution, including the summary in Table 5.5, we use the *mean absolute error (MAE)* to rank reduction methods. Mean absolute error is defined as

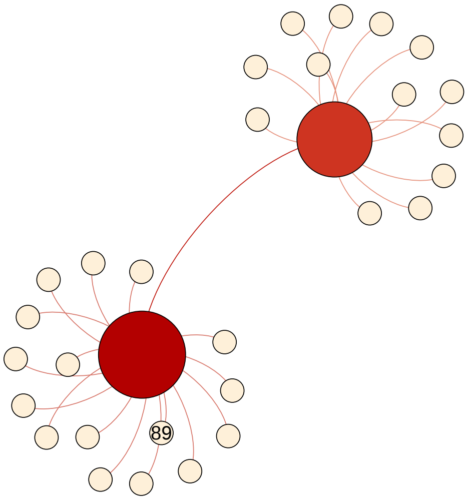


Figure 5.6: The inferred topology using AS 89 as a vantage point in the third round of probing.

$$MAE = \frac{1}{n}\sum_{i}|m_i - I_i|$$

where n is the total number of points compared, m_i is the reduction method data point, and I_i is the Internet data point value.

5.2.1 RouteViews Data Set: 2001–1998

We first aim to replicate the results of the original graph reduction experiments in [27]. This is a useful endeavor, as it demonstrates the reproducibility of the Krishnamurthy *et al.* results, which have not been reexamined in the literature since publication in 2007. Further, though we made contact with one of the authors, we were unable to obtain the source code used to produce the results of [27] – we make our source code available in order to aid future research in Internet graph sampling work. To begin, we compile a set inferred of Internet instances, beginning with the 7 May 2001 instance, and ending with the instance observed on 24 January 1998 from the RouteViews archive. We select these dates to mirror the dates selected by the original Krishnamurthy *et al.* study. Finally, RouteViews data is

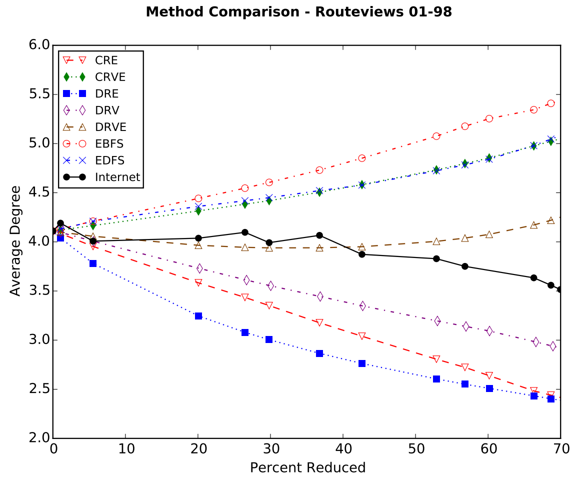


Figure 5.7: Average degree performance of non-DHYB methods – RouteViews 2001–1998

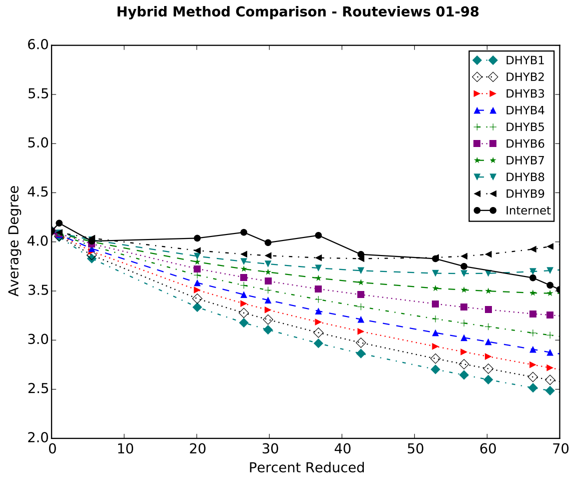


Figure 5.8: Average degree performance of DHYB methods – RouteViews 2001–1998

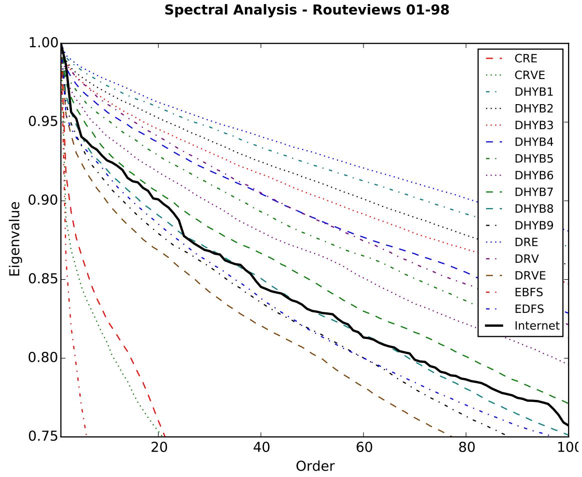


Figure 5.9: Spectra of reduction methods versus 24 January 1998 Internet graph – RouteViews 2001–1998

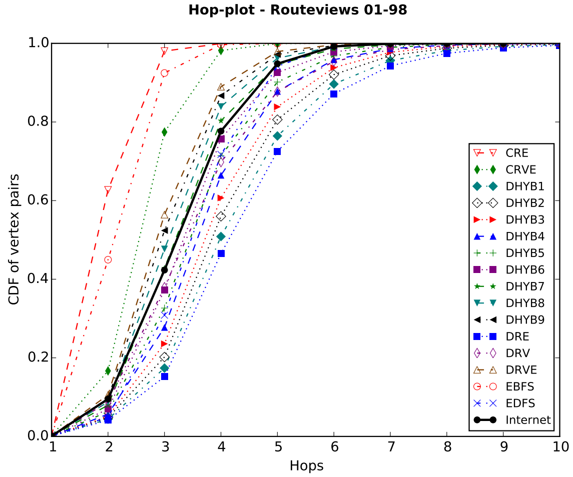


Figure 5.10: Hop plot of reduction methods versus 24 January 1998 Internet graph – RouteViews 2001–1998

released in RIB snapshots in `show ip route` format; these files must be parsed to create an AS-level Internet graph.

We note here that there is a choice to be made in how to measure the success or accuracy with which a reduced graph achieves the characteristics we find desirable. First, we can compare the reduced graph properties against the original graph instance – thus quantifying how well the reduced graph represents the original with fewer nodes. The second method is

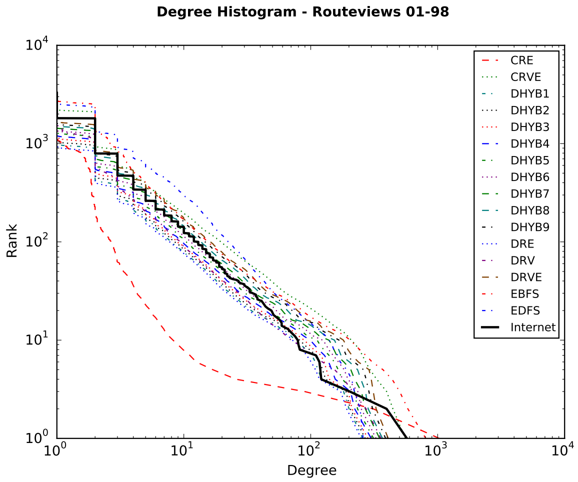


Figure 5.11: Degree histogram – RouteViews 2001–1998

to compare the reduced graph against an Internet graph *of the same order*, thereby judging how well the reduced graph represents the "*reverse-evolution*" of the Internet [27]. We, as with Krishnamurthy *et al.*, choose the second method, in which we compare our reduced graphs to an Internet instance with an equivalent AS count.

In our work, the 24 January 1998 graph serves as a target for each reduction to reach in terms of graph order. Further, because this graph is a known Internet instance, it also functions as a baseline to compare our reductions against. Pursuant to our goal of validating [27], we adopt their methodology of performing incremental reductions between initial Internet instance and the endpoint for the reduction. This allows us to examine the performance of the reduction algorithms as a function of total reduction percentage and permits us to find points at which algorithms diverge from the reverse-growth of the Internet.

Krishnamurthy *et al.* do not state which intermediate points were chosen in their study; we therefore select our own points interspersed between the 7 May 2001 and 24 January 1998 graphs. Our choice of intermediate instances, while arbitrary, does not influence the end results of our study because overall performance of the reduction algorithms is based on performance at the reduction endpoint of 24 January 1998.

For intermediate points, we choose instances on the first of December, June, and March between the start and end points when possible, or as close to the first as is available. When this data is compiled, a 25% difference in nodes between the initial May 2001 instance and the first intermediate point was noticed; for this reason, we also include points from January, February, March, and April 2001 for a more uniform sampling of inferred Internet data. A summary is given in Table 5.1.

Graph Instance	# Nodes	# Edges	Avg. Deg.	% Reduction
5 May 01	10966	22536	4.11	0
19 Apr 01	10851	22736	4.19	1.05
1 Mar 01	10359	20757	4.01	5.54
1 Feb 01	8765	17694	4.04	20.07
1 Jan 01	8062	16515	4.10	26.48
1 Jun 00	7696	15362	3.99	29.82
1 Mar 00	6935	14100	4.07	36.76
4 Dec 99	6293	12186	3.87	42.61
1 Jun 99	5167	9891	3.83	52.88
1 Mar 99	4736	8882	3.75	56.81
1 Jun 98	3695	6714	3.63	66.30
1 Mar 98	3436	6114	3.56	68.67
24 Jan 98	3291	5784	3.52	69.99

Table 5.1: A summary of the RouteViews inferred graphs used for the 2001–1998 reduction. The graph reduced by sampling algorithms is the 5 May 2001 graph; the reduction ends when the reduced graph contains approximately 3291 nodes, the number of ASs in the 24 Jan 1998 graph.

Using the 7 May 2001 inferred topology as our starting point, we proceed to reduce by all methods described in Section 2.3.6, taking averages of 50 different random seeds in all metrics examined due to the non-deterministic nature of all reduction methods. The number of random seeds averaged is chosen to match the work of Krishnamurthy *et al.*. When our reductions have reached a graph of order equivalent to one of our intermediate Internet instances, we capture this graph before continuing on to reduce it to a graph of approximately 3,291 nodes.

We first examine the average degree of the reduction algorithms as a function of percent of the initial Internet instance reduced. This is plotted against the Internet average degree as a function of the percent size difference between the intermediate points, and the 7 May 2001 graph. Because the percent difference of the Internet graphs is monotonically increasing with the time difference from 7 May 2001 for our intermediate points, the Internet line is can be viewed as the average degree over time, with the leftmost point representing the initial Internet instance and rightmost the reduction endpoint. In Figure 5.7, we observe results that are qualitatively the same as those observed in [27]. None of the non-DHYB methods perform particularly well at the end reduction point, though DRVE visually tracks the Internet line somewhat closely through 50% reduction in nodes, and is the closest to the average degree of the Internet at the reduction endpoint. At a node count equivalent to the 24 January 1998 Internet instance, DRVE has an average degree of 4.25, compared with the Internet average degree of 3.52.

When we examine the performance of the DHYB methods on this data set, we observe a slight divergence from the conclusions drawn by the authors of [27]. While Krishnamurthy *et al.* find that the performance of DHYB-0.8 most closely matches the Internet's average degree at 24 January 1998 instance, we find that DHYB-0.7 is even closer. Our results find DHYB-0.8 to have an average degree of 3.73, compared to DHYB-0.7 with 3.47. The average degree of the 24 January 1998 instance is 3.52. Despite our disagreement in terms of best reduction method for average degree, the difference between DHYB-0.7 and DHYB-0.8 is small, and may be attributable to variation due to the probabilistic nature of the reduction algorithms; however, Krishnamurthy *et al.* do not list DHYB-0.7 at any point in their results, making a determination difficult, if not impossible. The DHYB methods plotted with the Internet's average degree line is shown in Figure 5.8.

Next, we turn to an analysis of the spectra of the 24 January 1998 Internet instance, and compare these to the average of the spectral behavior of our reduction methods of equivalent order. As in [27], we examine the 100 largest eigenvalues of the normalized adjacency matrix of these graphs. These spectra provide an insight into the amount of clustering that occurs within these networks, as described in [35].

In Figure 5.9, we see that our average of DHYB-0.8 reduced graphs performs well through eigenvalues of order 25-75, with some divergence from the Internet's spectra among eigenvalues of order 5-25 and the higher ordered eigenvalues above 75. DHYB-0.8 has the lowest MAE of any of the reduction algorithms; as such, we declare it the winner. While finding DHYB-0.8 to be the best reduction method in terms of spectral behavior matches the findings of [27], we see several differences between the analyses. First, a cursory look at the data presented in the prior work shows our reduction methods plots to be far smoother than those in [27]. Second, prior work shows DHYB-0.8-reduced eigenvalues to be consistently higher than the Internet's, while our work shows them to be frequently lower. Based on an analysis of individual data points of our work, we believe that the authors of [27] have hand picked a particular reduction seed, rather than plotting the average all of the reduced graphs for a particular method.

Our next performance evaluation is *hop-plot*, which is defined as the CDF of the number of pairs of nodes in a graph reachable within k or fewer "hops" along a geodesic. As with spectral analysis, we look only at the average of the reduction methods at the most reduced point and compare to the Internet graph's earliest point, 24 January 1998.

Several methods perform well as measured by the *hop-plot* metric, as shown in Figure 5.10. DHYB-0.7 performs best at $k = 3$; differing by only .5% from the Internet's *hop-plot* line. At $k = 4$, we find that DHYB-0.6 offers the best performance, with 75.7% of reachable node-pairs compared to the Internet instance's 77.7%. DHYB-0.7 is again the best reduction method at $k = 5$, at which point 94.6% of node-pairs are reachable contrasted with the Internet graph's 94.8%. Prior work states that DHYB-0.8 is the best reduction method in terms of *hop-plot*; we find that DHYB-0.7 performs the best, as it has the lowest MAE of any reduction method. As with average degree, our difference in best method compared with Krishnamurthy *et al.* could be due to the probabilistic nature of the reduction algorithms; however, they do not mention DHYB-0.7 in their work.

Finally, we examine the degree distribution of the Internet inferred on 24 January 1998 as compared to the average degree distributions of the reduced graphs. This data appears in Figure 5.11.

While it is difficult to declare an absolute "winner" in terms of replicating the degree distribution of the Internet, we note here that several methods perform well at various points. First, CRVE comes the closest to matching the degree of the highest degree vertex of the Internet plot. It performs poorly throughout the rest of the vertices, however, producing consistently higher degrees than the Internet plot. Of the other methods, DHYB-$0.5, 0.6, 0.7$ all match the Internet degree distribution closely through the 10-100-degree vertex range. By MAE, DHYB-0.7 is the reduction method that most closely follows the Internet curve.

Because this metric is not studied in [27], we have no baseline against which to compare our results.

In summary, our results largely validate our implementation of our reduction algorithms. The shapes of our plots are effectively the same as those in [27], and the results, while not identical, are within the bounds of what can be considered reasonable given the randomness inherent in the reduction methods. Our average degree and *hop-plot* best methods were DHYB-0.7, compared to DHYB-0.8 found by Krishnamurthy *et al.* and we agree that DHYB-0.8 is the method that most closely matches the 24 January 1999 Internet spectra. In this source-period, we contribute the addition of the degree distribution analysis, and find that the best method agrees with our other results.

5.2.2 RouteViews Data Set: 2014–1998

Our novel work begins with considering RouteViews BGP RIB data over a longer period of time. In this source-period, our most recent Internet instance is from 1 December 2014 - an inferred AS-level graph of order 49,185 and edge-set cardinality 107,517, for an average degree of 4.37. Our reduction end point is the order of the Internet instance inferred on 1 January 1998, and intermediate instances are evenly distributed on the first of June and first of December for all years available. Our reduction end point is a graph consisting of 3,211 vertices and 5,611 edges, which is a node reduction of approximately 93.5%. This is a much larger reduction by absolute number of nodes and percentage than the Krishnamurthy *et al.* work; the original source-period represents an over 8,000 node reduction, or about 70%. The sources of our data for this source-period are summarized in Table 5.2.

We consider the same metrics over this source-period as we did in our reproduction of the original work.

In this second source-period, plotted in 5.12, we again see that none of the non-DHYB methods track the Internet's average degree plot well, and that none come close to arriving at the most reduced instance's average degree of approximately 3.6. The best performing non-DHYB method, DRV, has an average degree difference of approximately 1 from the Internet at the most reduced point; DRVE, the closest non-DHYB method in the Krishnamurthy *et al.* source-period, has an average degree of 7.92 – more than double that of the Internet at the most reduced point. DRVE tracks the Internet curve the most closely until

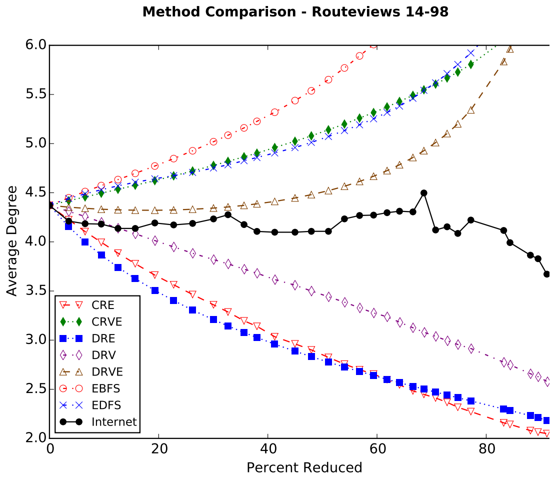


Figure 5.12: Average degree performance of non-DHYB methods – RouteViews 2014–1998

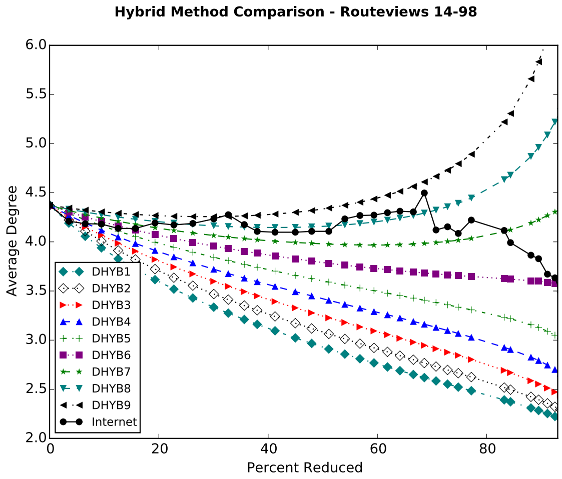


Figure 5.13: Average degree performance of DHYB methods – RouteViews 2014–1998

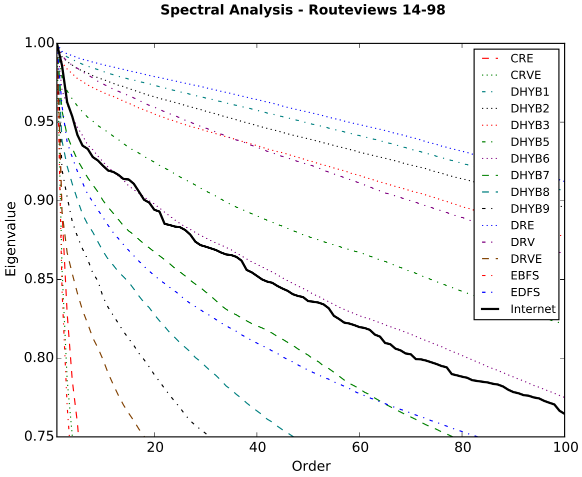


Figure 5.14: Spectra of reduction methods versus 1 January 1998 Internet graph – RouteViews 2014–1998

approximately a 70% reduction, at which point it reaches an inflection point and increases average degree at an increasing rate. This behavior was not observed in the first source-period, but the reduction endpoint was at approximately the same point at which DRVE diverged from the Internet in that reduction.

Of the DHYB methods, we see the best performance at the reduction endpoint from DHYB-0.6. The average of the DHYB-0.6 reduced average degrees is 3.56, which aligns closely

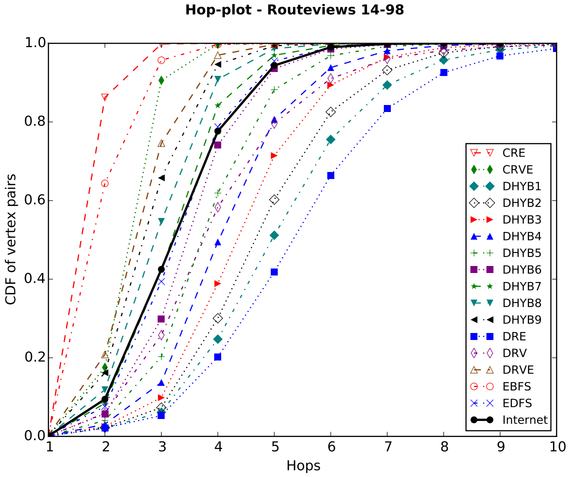


Figure 5.15: Hop-plot of reduction methods versus 1 January 1998 Internet graph – RouteViews 2014–1998

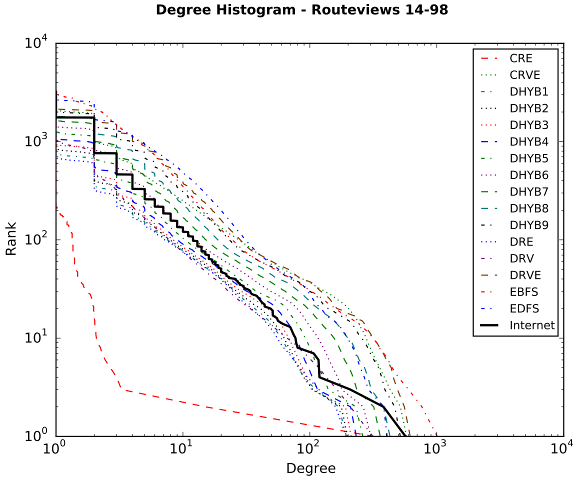


Figure 5.16: Degree histogram of reduction methods versus 1 January 1998 Internet graph – RouteViews 2014–1998

with the average degree of the 1 January 1998 instance, 3.49. None of the curves is a good fit overall, however. DHYB-0.7, our best performer in Section 5.2.1, differs in average degree by approximately 0.84 from the Internet at our endpoint. We note, however, that DHYB-0.8 and DHYB-0.7 track the Internet curve well through 50-70%, which is consistent with the results in the original source-period studied in [27]. The results of the DHYB reduction methods are shown in Figure 5.13.

Turning now to spectral behavior, we observe again that DHYB-0.6 outperforms all metrics in replicating the Internet's spectra, evaluated by MAE. DHYB-0.8, our winner in the original 2001–1998 source-period, comes in fifth by MAE, after DHYB-0.7, EDFS, and DHYB-0.5. The spectral plot for this data set is seen in Figure 5.14.

Next, we compare the *hop-plot* performance of the reduction methods. While, as in section 5.2.1, we see that both DHYB-0.6 and DHYB do well, when $k = 4$, we observe the performance of EDFS most closely matching the percentage of reachable pairs within 4 hops in the Internet instance. Indeed, EDFS is the closest to the Internet line by MAE, followed by DHYB-0.7, -0.6 and -0.8. DHYB-0.7 was the best method in our evaluation of *hop-plot*, while DHYB-0.8 was the winner in Krishnamurthy *et al.*'s study. Figure 5.15 displays our results.

Graph Instance	# Nodes	# Edges	Avg. Deg.	% Reduction	Graph Instance	# Nodes	# Edges	Avg Deg.	% Reduction
1 Dec 14	49185	107517	4.37	0	1 Dec 05	21246	45343	4.27	56.80
1 Jun 14	47449	99904	4.21	3.53	1 Jun 05	19984	42688	4.27	59.37
1 Dec 13	45974	96192	4.18	6.53	1 Dec 04	18769	40325	4.30	61.84
1 Jun 13	44517	93062	4.18	9.49	1 Jun 04	17647	38046	4.31	64.12
1 Dec 12	43011	88961	4.14	12.55	1 Dec 03	16439	35387	4.31	66.58
1 Jun 12	41457	85745	4.14	15.71	1 Jun 03	15450	34744	4.50	68.59
1 Dec 11	39693	83204	4.19	19.30	1 Dec 02	14397	29665	4.12	70.73
1 Jun 11	37996	79269	4.17	22.75	1 Jun 02	13361	27747	4.15	72.84
1 Dec 10	36289	75985	4.19	26.22	1 Dec 01	12396	25325	4.09	74.80
1 Jun 10	34406	72832	4.23	30.05	1 Jun 01	11219	23682	4.22	77.19
1 Dec 09	33100	70771	4.28	32.70	1 Dec 00	8260	17000	4.12	83.21
1 Jun 09	31655	66096	4.18	35.64	1 Jun 00	7696	15362	3.99	84.35
1 Dec 08	30492	62609	4.11	38.01	1 Oct 99	5854	11311	3.86	88.10
1 Jun 08	28890	59198	4.10	41.26	1 Jun 99	5167	9891	3.83	89.49
1 Dec 07	27064	55456	4.10	44.98	1 Dec 98	4369	8020	3.67	91.12
1 Jun 07	25591	52554	4.11	47.97	1 Jun 98	3695	6714	3.63	92.49
1 Dec 06	24051	49393	4.11	51.10	1 Jan 98	3211	5611	3.49	93.47
1 Jun 06	22607	47858	4.23	54.04					

Table 5.2: A summary of the RouteViews inferred graphs used for the 2014–1998 reduction. The graph reduced by sampling algorithms is the 1 Dec 14 graph; the reduction ends when the reduced graph contains approximately 3211 nodes, the number of ASs in the 1 Jan 1998 graph.

Lastly, for the more modern RouteViews data set, we again examine the results of the degree distributions produced by the various graph reduction methods in Figure 5.16. Of interest in this plot is that again CRVE performs well only at the highest degree vertex; similar to our results in Section 5.2.1, we again see that several DHYB reductions perform well throughout the distribution of vertices between 10 and 100. In this case, DHYB-0.4,-0.5, and -0.6 most closely follow the Internet degree distribution. By MAE, DHYB-0.5 is the winner, with DHYB-0.6 and DHYB-0.4 close behind.

5.2.3 CAIDA Data Set: 2001–1998

In this section, we turn our attention to the data collected by CAIDA over the same timeframe as the initial study done by the authors of [27]. This is the first source-period we analyze that was obtained using CAIDA-derived data, and is an AS-level Internet topology that was obtained in a distinct manner from RouteViews. CAIDA compiled data was not studied in [27]. Our initial instance comes from 1 May 2001, the closest data point compiled by CAIDA to the original analysis in Section 5.2.1. This graph consists of 11,045 nodes and 24,485 edges, for an average degree of approximately 4.43. It should be noted that while the order of the graph is similar to that obtained from the RouteViews data, the edge-set cardinality of the graph is larger by about 2000 edges. The reduction end point for this data set is the inferred Internet instance from 1 January 1998. The reduction end point

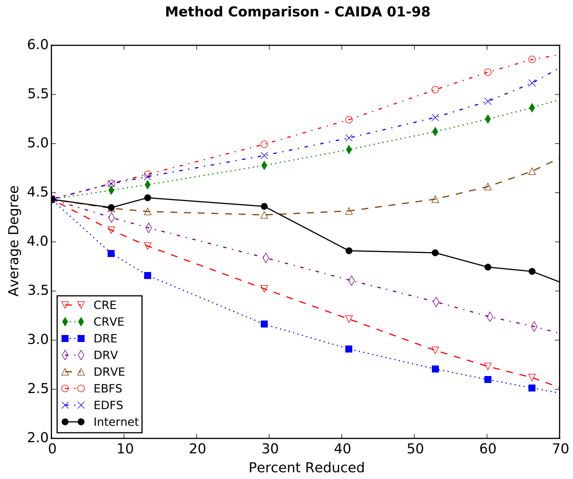


Figure 5.17: Average degree performance of non-DHYB methods – CAIDA 2001–1998

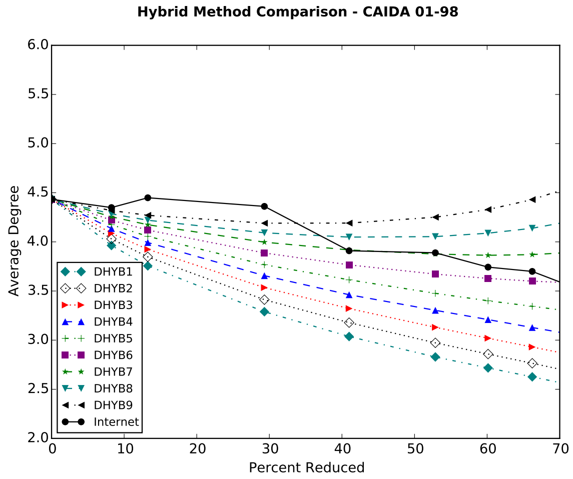


Figure 5.18: Average degree performance of DHYB methods – CAIDA 2001–1998

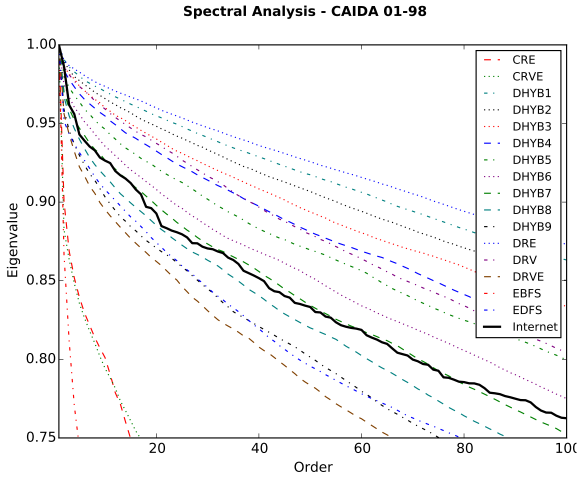


Figure 5.19: Spectra of reduction methods versus 1 January 1998 Internet graph – CAIDA 2001–1998

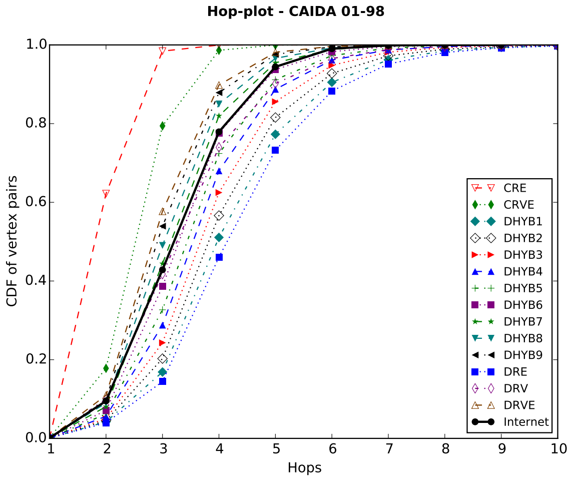


Figure 5.20: Hop plot of reduction methods versus 1 January 1998 Internet graph – CAIDA 2001–1998

graph contains 3,233 nodes and 5,773 edges, with an average degree of about 3.57. As in the prior data sets, we chose intermediate data points from the first of June and December between the start and end of our reduction time frame if possible; if not, an inferred graph close to those dates was selected. Table 5.3 provides a summary of the CAIDA inferred graphs that were used.

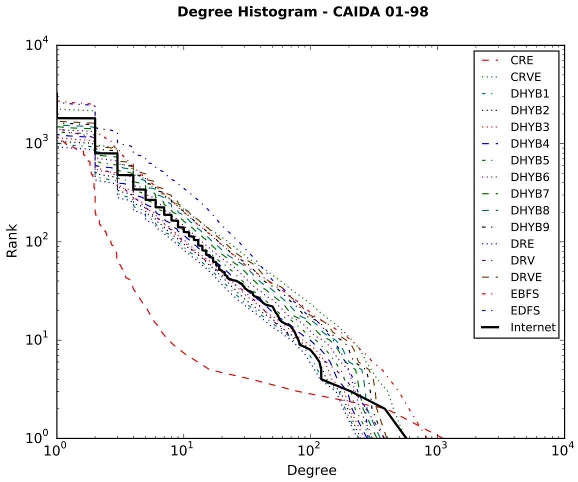


Figure 5.21: Degree histogram of reduction methods versus 1 January 1998 Internet graph – CAIDA 2001–1998

Graph Instance	# Nodes	# Edges	Avg. Deg.	% Reduction
1 May 01	11045	24484	4.43	0
1 Feb 01	10134	22037	4.35	8.25
1 Dec 00	9581	21312	4.45	13.25
1 Jun 00	7807	17024	4.36	29.32
1 Jan 00	6518	12741	3.91	40.99
1 Jun 99	5206	10123	3.89	52.87
1 Dec 98	4404	8242	3.74	60.13
1 Jun 98	3732	6903	3.70	66.21
1 Jan 98	3233	5773	3.57	70.73

Table 5.3: A summary of the CAIDA inferred graphs used for the 2001–1998 reduction. The graph reduced by sampling algorithms is the 1 May 01 graph; the reduction ends when the reduced graph contains approximately 3233 nodes, the number of ASs in the 1 Jan 1998 graph.

In this source-period, our average degree experiments show that none of the non-DHYB reduction methods perform well; none approximate the Internet graph from January 1998 well. DRV is the best non-DHYB reduction algorithm for this source-period – its average degree at the most reduced point differs from the Internet graph by about 0.5. DRVE, the winner of the non-DHYB methods in the original source-period, comes in second, differing by about 1.3. As with prior source-periods, we note that DRVE tracks the Internet plot fairly well through 30% reduction, however. See Figure 5.17.

Of the DHYB reduction methods, we find that DHYB-0.6 performs the best at the most reduced point. Again, none of the curves fit the Internet's plot throughout the whole time period considered, however. Figure 5.18 displays these findings. This is similar to our results in the original RouteViews data set in Section 5.2.1, in which we found DHYB-0.7 to be the winner of the average degree comparison, and the same as the more modern RouteViews data set in Section 5.2.2.

When evaluating spectral behavior, DHYB-0.7 is the clear winner by MAE. This reduction method tracks the sorted eigenvalues extremely well throughout the majority of the top 100 considered. This is the first data set in which DHYB-0.7 has performed the best in this metric, but it is between the DHYB-0.8 and DHYB-0.6 found in Sections 5.2.1 and 5.2.2, respectively. Our data is displayed in Figure 5.19.

Our analysis of *hop-plot* data shows several reduction methods track with the Internet data at various $k-$hop distances. At $k = 2$, DRV, DHYB-0.8 and DHYB-0.9 all fall within 6% of the Internet's data point of 0.095. As k increases, DHYB-0.6 and DHYB-0.7 bound the Internet line, with DHYB-0.6 offering marginally better performance. By MAE, DHYB-0.7 performs the best, as in our spectral analysis for this source-period. This can be seen in Figure 5.20.

Finally, we examine the Internet's degree distribution at the 1 January 1998 reduction end point versus the averaged degree distributions of our reduced graphs. For the third source-period, CRVE most closely matches the highest degree node in the Internet instance, though it tracks the rest of the Internet line poorly. Further, as in Section 5.2.4, we find that DHYB-0.4,-0.5 and -0.6 track the Internet's degree distribution most closely through the vertices of degree 10-100. By the MAE metric, DHYB-0.5 is the best performer overall. Our results are displayed in Figure 5.21.

5.2.4 CAIDA Data Set: 2014–1998

Lastly, we examine the CAIDA data set from 1 December 2014 to 1 January 1998. Our largest Internet instance is a graph comprised of 46,177 nodes, with 177,391 edges for an average degree of 7.68. This average degree is the first major difference between this data set and those studied earlier in this section; it is far higher than the roughly 3.5 of the other data sets. This graph is then reduced, using our reduction methods, to graphs that

Graph Instance	# Nodes	# Edges	Avg. Deg.	% Reduction	Graph Instance	# Nodes	# Edges	Avg Deg.	% Reduction
1 Dec 14	46177	177391	7.68	0	1 Dec 05	21298	55351	5.20	53.88
1 Jun 14	45767	167156	7.30	0.89	1 Jun 05	20106	51327	5.11	56.46
1 Dec 13	45427	159049	7.00	1.62	1 Dec 04	18827	48532	5.16	59.23
1 Jun 13	44611	151434	6.79	3.39	1 Jun 04	17662	44603	5.05	61.75
1 Dec 12	43109	137031	6.36	6.64	1 Dec 03	16470	41232	5.01	64.33
1 Jun 12	41580	128040	6.16	9.96	1 Jun 03	15488	37363	4.82	66.46
1 Dec 11	39902	121969	6.11	13.59	1 Dec 02	14370	30775	4.28	68.88
1 Jun 11	38162	115098	6.03	17.36	1 Jun 02	13381	29536	4.41	71.02
1 Dec 10	36396	103185	5.67	21.18	1 Dec 01	12468	27668	4.44	73.00
1 Jun 10	34832	100272	5.76	24.57	1 Jun 01	11286	24557	4.35	75.56
1 Dec 09	33339	95299	5.72	27.80	1 Dec 00	9581	21312	4.45	79.25
1 Jun 09	31778	89214	5.61	31.18	1 Jun 00	7807	17024	4.36	83.09
1 Dec 08	30404	85126	5.60	34.16	1 Oct 99	5882	11564	3.93	87.26
1 Jun 08	28683	79372	5.53	37.88	1 Jun 99	5206	10123	3.89	88.73
1 Dec 07	27147	74710	5.50	41.21	1 Dec 98	4404	8242	3.74	90.46
1 Jun 07	25693	70097	5.46	44.36	1 Jun 98	3732	6903	3.70	91.92
1 Dec 06	24128	63807	5.29	47.75	1 Jan 98	3233	5773	3.57	93.00
1 Jun 06	22641	59670	5.27	50.97					

Table 5.4: A summary of the CAIDA inferred graphs used for the 2014–1998 reduction. The graph reduced by sampling algorithms is the 1 Dec 2014 graph; the reduction ends when the reduced graph contains approximately 3233 nodes, the number of ASs in the 1 Jan 1998 graph.

have a comparable number of nodes to the Internet instance from 1 January 1998, which has order 3,233 and edge-set cardinality of 5,773. This is the same reduction endpoint as Section 5.2.3; a summary of the graphs used in this source-period is given in Table 5.4.

In our examination of the average degree performance of non-DHYB methods, we see that DRV comes within about 10% of the average degree of the most reduced Internet instance - DRV with 3.97, and the final Internet instance with 3.57. This matches our other CAIDA source-period, but differs from the RouteViews Internet graphs in that DRVE is not the best performing non-DHYB method. Our non-DHYB data is shown in Figure 5.22.

In the DHYB methods, we see for the first time very different results. DHYB-0.1 and DHYB-0.2 are clearly the best reduction methods, and bound the January 1998 instance. With average degrees of 3.06 and 4.06, respectively, DHYB-0.1 and DHYB-0.2 do not match the end point's average degree as well as DRV. See Figure 5.23.

In our spectral analysis, plotted in Figure 5.24, the graphs reduced by the DHYB-0.2 reduction method exhibit the most similarity to the January 1998 Internet instance by MAE. Though the curve is slightly higher than that of the Internet for most of the 100 largest eigenvalues, the DHYB-0.2 slope easily bests all other methods. Further, in Figure 5.24, we note that all but four of the reduction methods (DHYB-0.1, DHYB-0.2, DRE, and

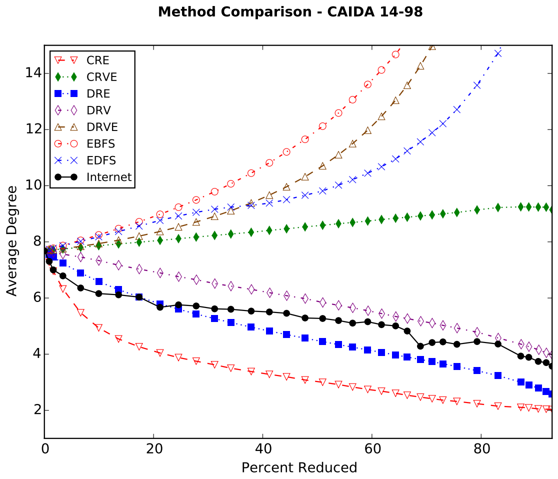


Figure 5.22: Average degree performance of non-DHYB methods – CAIDA 2014–1998

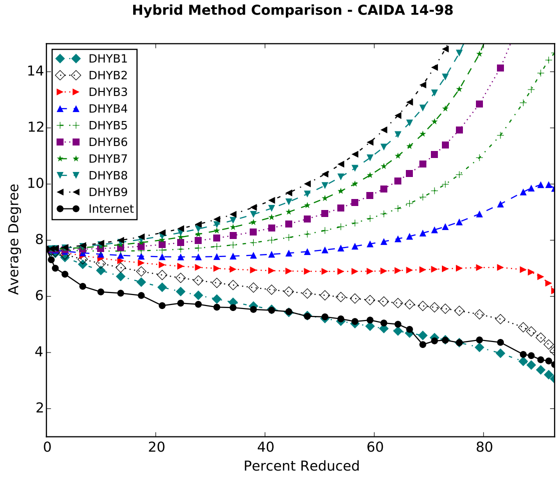


Figure 5.23: Average degree of DHYB methods – CAIDA 2014–1998

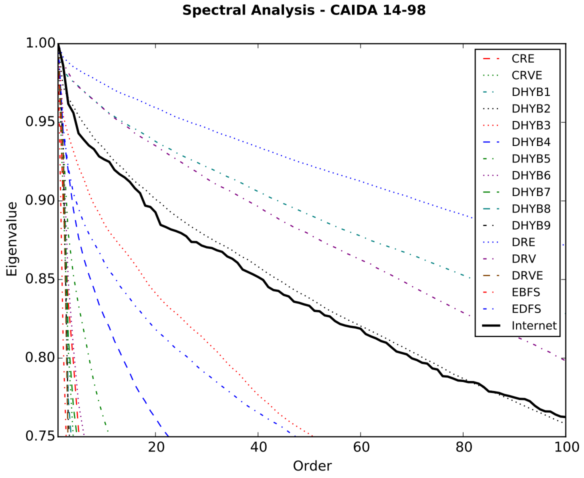


Figure 5.24: Spectra of reduction methods – CAIDA 2014–1998

DRV) produce spectral plots with normalized eigenvalues that quickly decrease below the .75 mark. This indicates that the amount of clustering is dramatically less than the Internet graph at the most reduced point for these reduction methods.

The *hop-plot* results for the extended CAIDA data set indicate interesting results as well, as seen in Figure 5.25. DRV most closely matches the CDF of node-pairs reachable within k-hops through $k = 5$, with EDFS bounding the Internet curve above. By MAE, DHYB-0.3

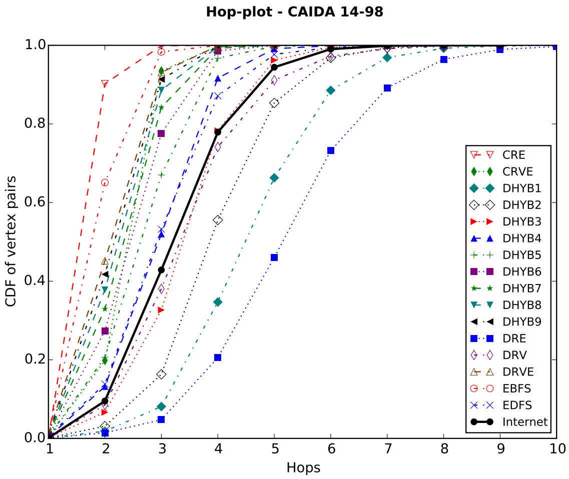


Figure 5.25: Hop-plot of reduction methods – CAIDA 2014–1998

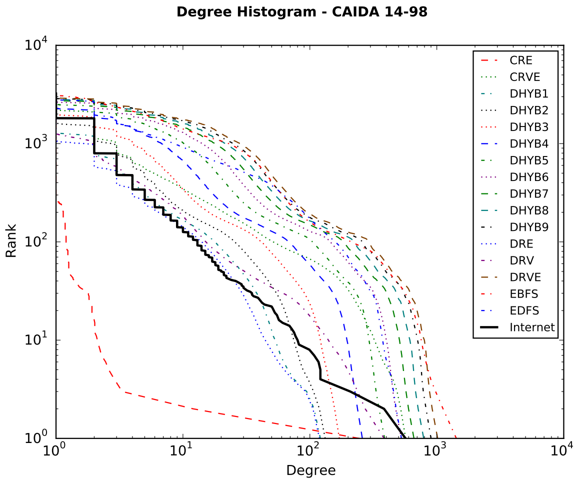


Figure 5.26: Degree distribution of reduction – CAIDA 2014–1998

is the best method in *hop-plot*, followed by DRV. This is in stark contrast to the CAIDA source-period, in which DHYB-0.7 matches the Internet *hop-plot* most closely.

Lastly, we examine the degree distributions of our reduced graphs compared with the Internet instance end point. Again in Figure 5.26, we see results that significantly differ from those seen in previous sections. No method performs particularly well; CRVE again best matches the highest degree vertex, and DRE and DRV follow the distribution of vertices of degree 30 and less fairly closely, but no method can realistically be called effective. The closest method by MAE is DHYB-0.1, followed by DRE and DRV. This is the first metric in any source period in which DRE performs significantly above average.

5.2.5 KDD and KKD Results

We first note that we have no average degree plots to present; this is because, as noted earlier, KDD and KKD take the target edge count as an input parameter. Requiring an edge count allows us to reach our target average degree precisely.

Next, we present the plots of KDD and KKD reduced graph spectra against those of the reduction end points across both data sets and time periods. We note here that in both data sets, the longer timeframe from 2014 to 1998 exhibits better performance than in the shorter timeframe from 2001 to 1998. This is likely the case because more *k*-cores could

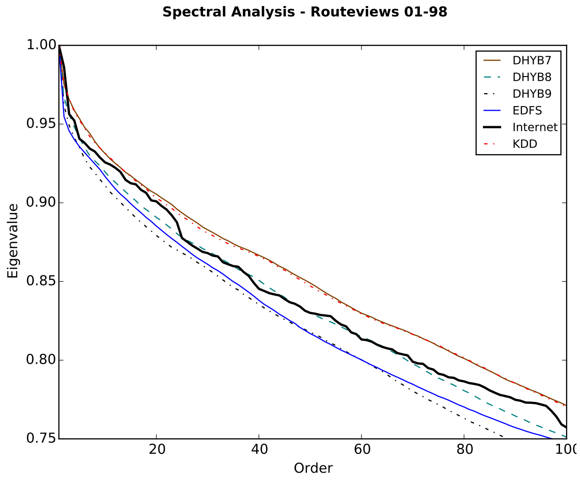


Figure 5.27: KDD is the second best reduction method in terms of spectral analysis.

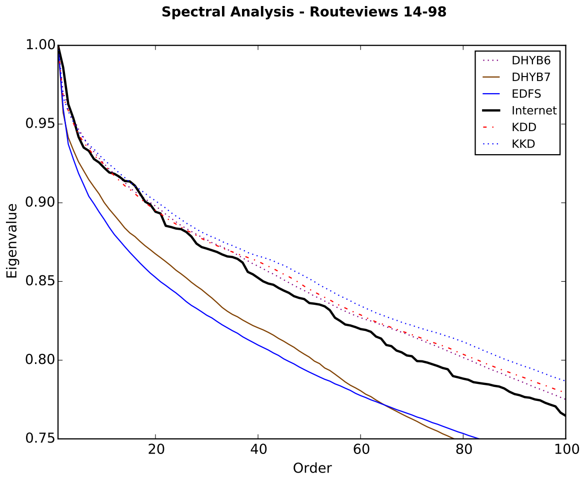


Figure 5.28: KDD and KKD are the second and third best spectral performers by MAE.

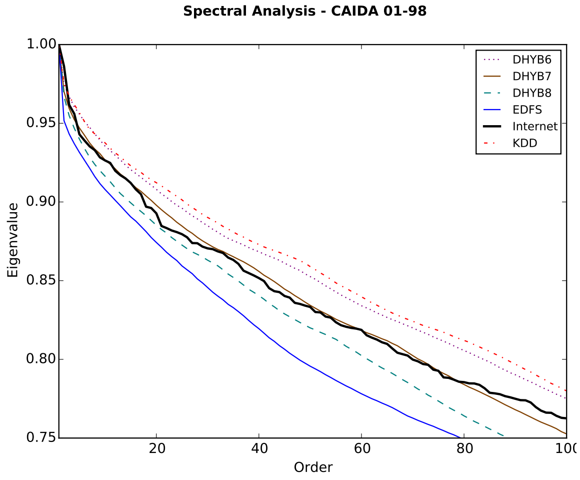


Figure 5.29: Spectral analysis shows KDD to be the fourth best performing reduction method.

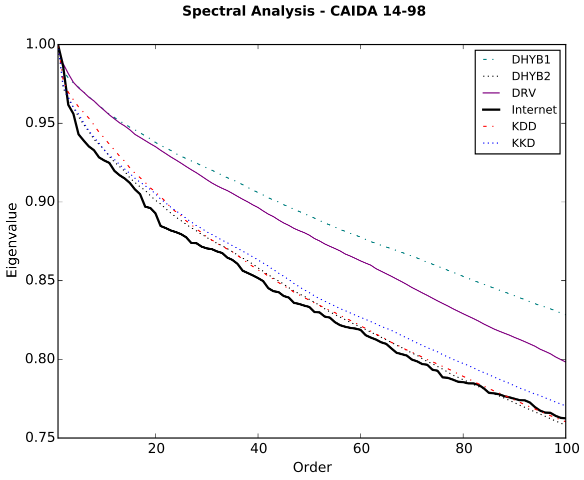


Figure 5.30: KDD and KKD are the second and third best performing reduction methods in spectral analysis.

be taken in the data sets over the larger time period – in both the CAIDA and RouteViews data sets beginning in 2014, we stopped our k-core reduction process at $k = 8$; in the data sets beginning in 2001, however, we had to stop at $k = 2$, and reduce much further using the k-deletion or DRVE phases. The ability to obtain k-cores for higher k values causes the more central ASs to be separated from the lower degree, lower clustering periphery. The difference in gaps between the Internet curve and the KKD and KDD curves in Figures 5.27

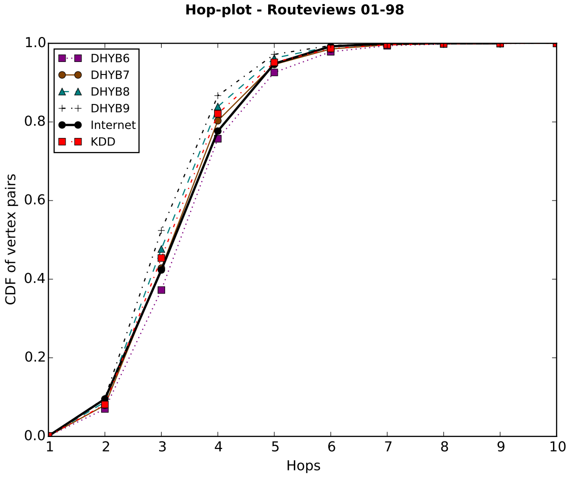


Figure 5.31: Five best reduction methods for hop-plot. KDD is the second best performer.

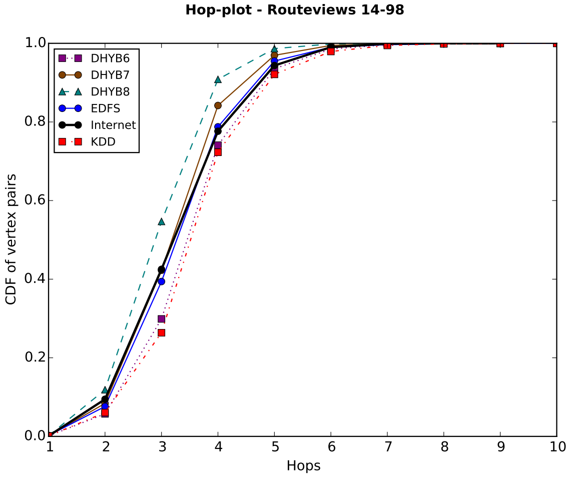


Figure 5.32: Plot of five best reduction methods for hop-plot; KDD is the fourth best.

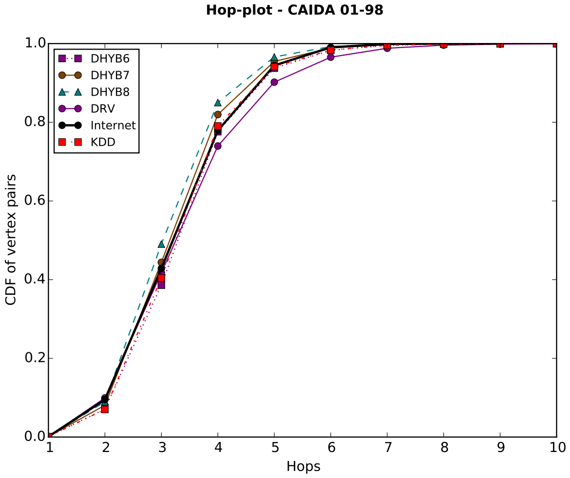


Figure 5.33: Hop-plot of five best reduction methods. KDD most closely matches the Internet plot.

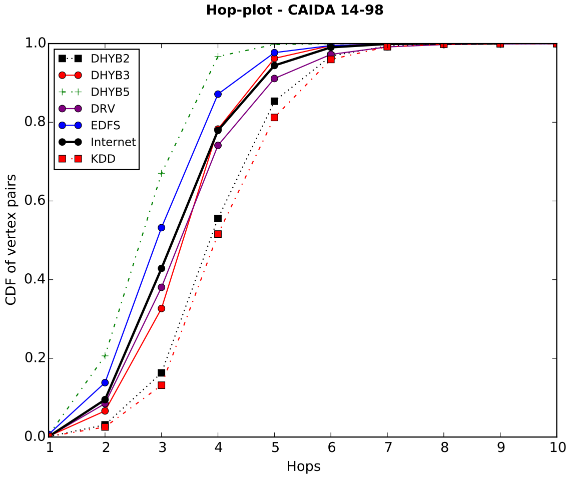


Figure 5.34: KDD is the seventh closest reduction method, or slightly better than average.

and 5.28 and Figures 5.29 and 5.30 demonstrate this phenomenon. For these reasons, we believe that our reduction methods KDD and KKD are better suited to initial Internet instances with either a higher degree, or over a longer time period, using more modern data.

Further, we see the spectra of KDD outperform KKD in all four plots; the spectra of KKD are higher on average than those of both the Internet and KDD. We believe this is caused by the removal of only the *k*-degree vertices and their incident edges in the second step

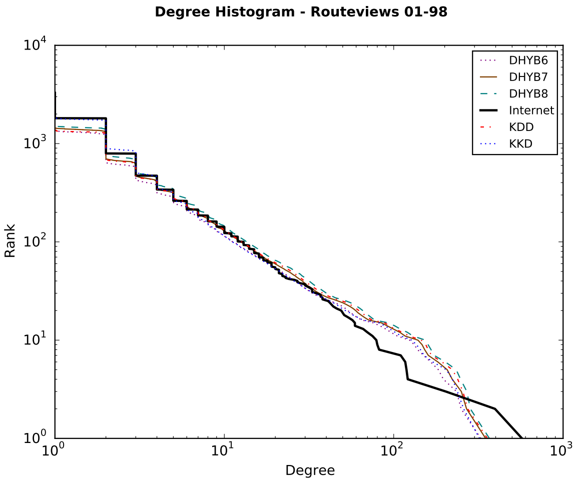


Figure 5.35: KKD most closely matches the Internet degree distribution by MAE. KDD comes in fourth.

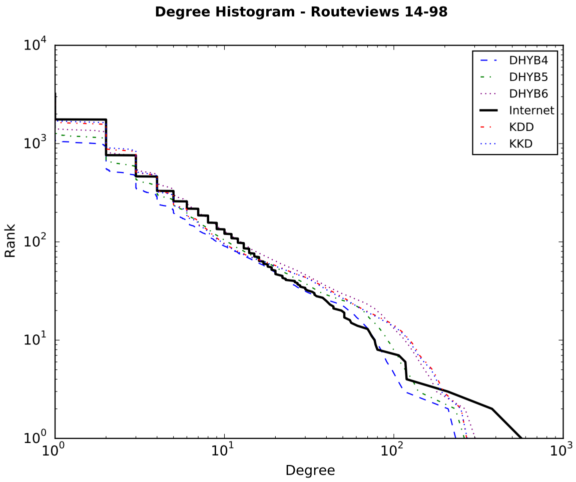


Figure 5.36: KKD and KDD are the first and second most closely matching reduction methods, respectively.

of KKD; in doing so, we tend to preserve the largest, most connected clusters within the graph. In addition to being the better performer of our two novel reduction methods, we find that KDD does well across all time periods. In three of the four (RouteViews 2014–1998, RouteViews 2001–1998, and CAIDA 2014–1998), KDD is the second best reduction method using the MAE between its plot and the Internet plot for comparison. In the fourth, CAIDA 2001–1998, it is the fourth closest reduction method. We believe these results make KDD a valuable tool for reducing Internet topologies across varied data sets and time periods, especially when the accuracy of spectral properties is important. For example, KDD is the second best performing method in both of the 2014–1998 data sets, despite the top performers differing in their DHYB probability value by 40%.

In Figures 5.31, 5.32, 5.33, and 5.33, we show the *hop-plot* performance of KDD and KKD. Unlike our spectral results, we see above-average performance for KDD and KKD in the data sets from 2001–1998, while our plots do not follow the Internet curve nearly as well in the 2014–1998 period. In the CAIDA data from the shorter time period, KDD is the best performing reduction method, while in the 2001–1998 RouteViews comparison, it is the second closest reduction method to the Internet plot. The longer timeframes decrease the utility of our new *k*-core based reduction methods; in the 2014–1998 RouteViews study, it is the fourth best method, while it comes in seventh in the CAIDA 2014–1998 time period.

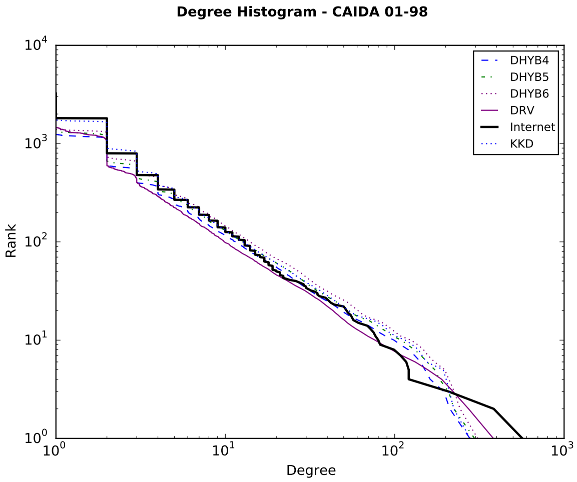


Figure 5.37: KKD best follows the Internet degree histogram.

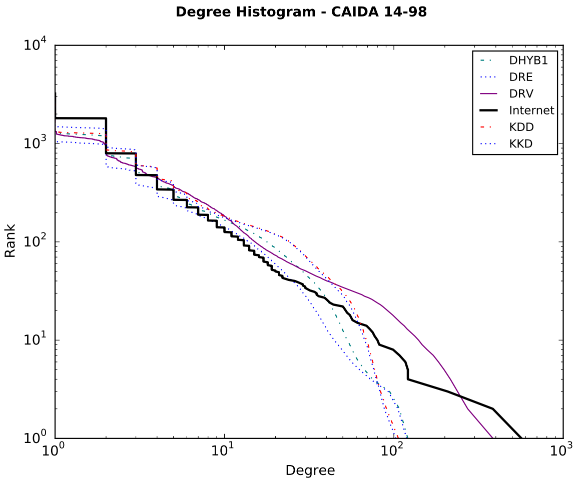


Figure 5.38: KKD and KDD are the fourth and fifth best performers, respectively.

Across all time periods and data sources, we note that KDD performs better than KKD; in no case does KKD enter the top five most closely-matching methods.

Finally, in Figures 5.35, 5.36, 5.37, and 5.38, we show the degree histograms of KDD and KKD reduced graphs against the Internet reduction end point. In both data sets and time periods, our reduction methods produce highest-degree vertices of several hundred less than in the Internet instance. This is somewhat common across many of the reduction methods, as we have seen previously. In three of these plots, Figures 5.35, 5.36, 5.37, we observe above-average performance from the 100-1 degree range from our reduction methods, KDD in particular. In Figure 5.38, however, both KKD and KDD reduction methods exhibit a tendency to create vertices of higher degree than the Internet through the 20-110 rank range. Unlike the metrics examined previously, KKD follows the Internet curve slightly better than KDD throughout much of the degree range; KDD tends to produce a slightly higher degree high-degree vertex, however. In general, KDD and KKD perform exceptionally well in terms of matching the degree distribution of the target Internet graph. In three of the four data sets (CAIDA 2001–1998 and both RouteViews time periods), KKD is the best-performing metric by the MAE. In the fourth, CAIDA 2014–1998, KKD and KDD are the fourth and fifth plots most closely following the Internet curve, respectively. Again, we believe this points to the utility of our novel reduction methods. The best re-

	RV 01-98	**RV 14-98**	**CAIDA 01-98**	**CAIDA 14-98**
Avg. Deg	DHYB-0.7	DHYB-0.6	DHYB-0.6	DRV DHYB-0.1,0.2
Spectral	DHYB-0.8 KDD	DHYB-0.6 KDD	DHYB-0.7 DHYB-0.8	DHYB-0.2 KDD
Hop Plot	DHYB-0.7 KDD	EDFS DHYB-0.7	KDD DHYB-0.7	DHYB-0.3 DRV
Deg.Dist.	KKD DHYB-0.7	KKD,KDD DHYB-0.5	KKD DHYB-0.5	DHYB-0.1 DRE

Table 5.5: Summary of best reduction methods per source-period considered

duction methods studied by prior work are DHYB methods whose probability value range from 0.1 to 0.7 across the data sets.

In conclusion, we observe that KDD and KKD tend to perform well across all source-periods we consider, with increased benefits occuring over the longer timeframes. While these methods are infrequently the best reduction algorithm, they are within the top five reduction methods considered in every metric, with the lone exception of *hop-plot* in the CAIDA data from 2014–1998, in which KDD remains above average. We believe that this indicates our methods offer a better technique for sampling large Internet topologies than using purely probabilistic reduction algorithms, as they leverage a property of these graphs. Table 5.5 summarizes the best performers across the four distinct source-periods considered.

5.3 Reduced Graphs Emulated on ERIK

The impetus behind sampling large Internet topologies in order to obtain smaller, representative graphs in Section 5.2 is to obtain topologies that are realistic, but able to be emulated or simulated. Beginning with a 2001 RouteViews Internet topology, we reduced it to an order of 300 nodes, and emulated this topology using ERIK. DHYB-0.7 was selected as the reduction method, as this was the sampling method that most closely matched the Internet graph characteristics at the reduction endpoint in Section 5.2 for this source-period. The results of this emulation are the subject of this section.

In order to successfully implement the BGP policy model outlined in Section 3.2.1, we made several modifications to ERIK when configured to constructively create tiered, hierarchical topologies. First, we categorize the ASs in our topology in the following manner:

1. The five ASs with the highest degree were selected to be Tier 1 nodes in our network.
2. Next, all degree-one ASs in the topology were categorized as customer ASs.
3. Finally, remaining nodes were chosen to be Tier 2 nodes, and given the BGP policy we defined for Tier 2 nodes in Section 3.2.1.

The resulting network created by this modification differs from the tiered model in several regards. These choices preclude any customer nodes from being *dual-homed*, though it does result in the creation of Tier 2 nodes that are *dual-homed* to Tier 1 ASs without customers themselves. This choice was made in order to accommodate situations in which the shortest path from a customer node to a Tier 1 node was greater than length 2, an impossibility in the tiered model. Several of these paths occur in the reduced topologies, and for this reason, reduction results in far more Tier 2 ASs than in our other emulation experiments.

The sampled topology we examine here consists of 5 Tier 1 ASs, 126 Tier 2 ASs, and 169 customer ASs, connected by 497 edges. In the first inference round before failures occur, we discover a complete AS-level topology from every AS vantage point, as before.

In our second round of inference probing, during link failures, paints a different picture than in Section 5.1. While link faults again induce routing table churn, and cause many ASs to go undiscovered during probing, this occurs less often in this topology. In Figure 5.39, we show the CDF of vantage point ASs by fraction of the topology that was missed during inference probing during the second probing round. Over half of the AS vantage points miss very little of the topology; approximately 5%, or 15 ASs, is the median AS count missed. Figure 5.40 breaks down the vantage points by tier; here, we see that Tier 1 ASs have a higher median fraction of the topology missed than Tier 2 or customer ASs. However, in this topology, we observe the same long tails seen in Section 5.1 for both Tier 2 and customer ASs. These tails terminate at approximately 50% of ASs missed, as opposed to the 90% not discovered in Section 5.1.

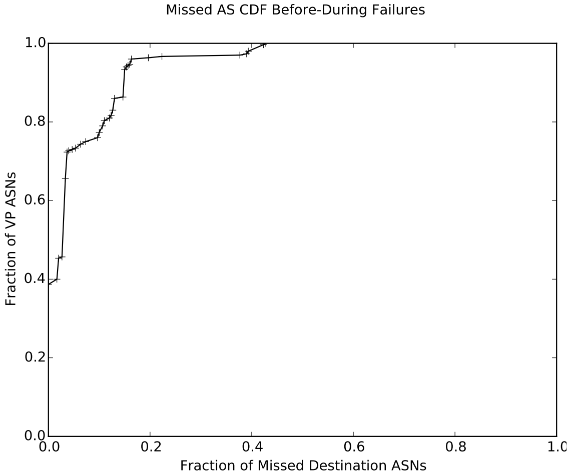


Figure 5.39: CDF of AS vantage points by fraction of missed ASNs during second probing round with failures.

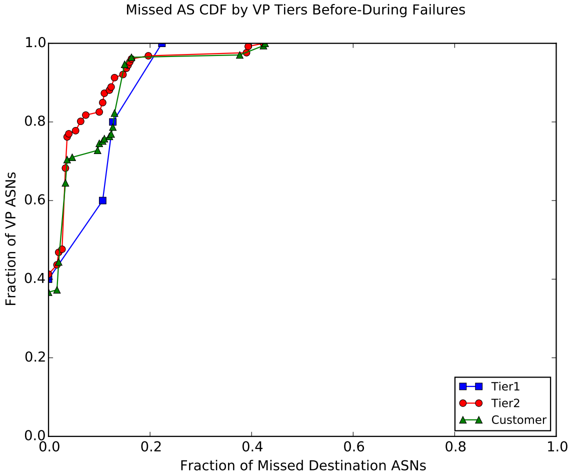


Figure 5.40: CDF of AS vantage points by fraction of missed ASNs during second probing round with failures, separated by vantage point AS tier.

The results of our third, final probing round after routing table reconvergence differ from the emulated topology studied in Section 5.1 as well. At each AS vantage point, we discover the entire topology once again, obviating any need for CDFs for this probing round.

In light of these results, we conclude that this topology is more robust to failures than the first topology examined in Section 5.1, despite having 179 fewer edges. We make this conclusion based on the number of ASs not discovered in the first topology during the second inference round, compared with the results of the second round of probing in the sampled topology. Figures 5.1 and 5.40 detail this difference in resilience to link failures. Further, we note that because all ASs were discovered from all vantage points in the final round of probing, the sampled topology contains no policy disconnections after the link failures have occurred.

In the next chapter, we conclude, and offer our suggestions for future work that builds on our methodology and analysis discussed previously.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6:
Conclusions and Future Work

In this thesis, we proposed and developed a tool, called ERIK, to automate:

1. The construction of network topological ground truth, or knowledge of the router/AS-level graph of a network, including routing policy and IP address assignment.
2. Creation and set-up tasks to emulate these network topologies.
3. Topology inference experimentation – exhaustive probing of the topology from each possible vantage point in the network.
4. Events, such as router-to-router link failures, that induce pathologies in topology inference.

Our methodology for creating this utility involves generating a network graph, according to a well-known or our own topology generation algorithm, and creating Cisco configuration files for each router, according to a predefined policy model. In addition, we create the series of *Dynamips* commands necessary to run the emulation. When the emulated topology has been initialized, we manage the inference of the network topology through an emulated Linux VM.

Chapter 5 details results obtained from exhaustively inferring an example topology from each vantage point in the network. We see a wide variance in the amount of topology that is discovered from each vantage point when the topology is probed during a set of link failures. Our results show that the dynamic nature of the wider Internet, with hardware and physical connections between ASs failing, can influence the outcome of inference probing significantly.

In addition, we undertake a reexamination of prior work in Internet graph sampling by Krishnamurthy *et al.*, and expand upon their work by considering more modern Internet topologies and Internet topological data from a different source. Our Internet graph sampling methodology mirrors that of [27]; we begin with an initial Internet instance from RouteViews or CAIDA, and sample this graph until we obtain a graph with a target number of nodes. We then compare graph properties such as average degree, adjacency matrix eigenvalues, and degree distribution to an Internet instance of the reduced graph's order.

We perform graph sampling on four data source/time period combinations, and see a wider variety of results than were obtained in prior work. We conclude that the probabilistic reduction methods studied in [27] are not sufficient to effectively sample the Internet graph across source-periods, and develop two of our own graph sampling methods, known as KDD and KKD, that involve taking successive *k*-cores of the Internet topology. We show that for several metrics, these reduction algorithms outperform the probabilistic algorithms across source-periods.

6.1 Future Work

In this section, we detail our suggestions for work that builds on our results and analysis.

Increase ERIK scale

At present, using a commodity server, we are able to emulate virtualized router topologies that contain several hundred routers with complex interconnections. We believe that with several interconnected servers, ERIK can be scaled up to emulate virtualized router topologies consisting of over a thousand routers by physically interconnecting these machines.

The primary obstacle to increasing the scale of ERIK is determining the underlying *Dynamips* commands and configurations necessary to achieve such an emulated topology over a distributed architecture. Further, as the emulated networks grow in scale and complexity, our models for generating these networks should be reevaluated to ensure we are generating topologies consistent with our goals.

Increasing the number of emulated routers could provide many benefits. Of them, the ability to emulate larger, more realistic networks is the most promising. AS-level Internet topology data began being collected by CAIDA and RouteViews when the Internet consisted of approximately 3,000 ASs; an increase of one order of magnitude would allow us to emulate the entire AS-level graph, abstracting each AS as a router. Further, increased scale allows for the construction of topologies that avoid this AS-to-router abstraction, as we discuss in the next topic.

Intra-AS Topology Emulation

In Chapter 5, we demonstrate the power of using ERIK to emulate AS-level topologies in which each AS is represented by a single emulated Cisco 7200 series router. This is an abstraction of the complex and dynamic intra-AS topology found in ASs on the Internet, which can consist of hundreds or thousands of routers. The decision to remove the internal AS structure and contract each AS to one router was made with the goal of creating the largest AS-level topologies in mind. However, with increased scale, ERIK could be used to design intra-AS topologies in addition to the inter-AS connections described in this thesis.

JunOS-Pure and Mixed Topologies

The results obtained in Chapter 5 using ERIK were collected from inferences of purely Cisco router topologies. We believe it would be useful to perform the same experimentation on the same network topologies in which each router runs Juniper's OS, JunOS, in order to determine whether similar results are obtained. For example, JunOS may handle BGP route updates in a different manner than Cisco IOS. Direct comparison to Cisco-pure topologies could easily be made using the same network generation model and the same random seed.

Barriers to emulating these types of topologies are practical – while *Dynamips* provides a software platform for emulating Cisco routers, there is no analog for JunOS. Thus, creating JunOS-pure or mixed topologies would require much more significant development of the behind-the-scenes infrastructure that *Dynamips* enables us to ignore.

Inferred Topology Sensitivity to BGP Load and Network Model

In Chapter 5, we examine the number of ASs that were not discovered during topology inference tests from all vantage points. In future studies, we propose studying the sensitivity of these undiscovered ASs to the BGP load injected into the topology by our Linux VM using `bird` to determine whether there is a correlation between the amount of churn in the emulated routers' RIB tables and the number of prefixes carried.

Additionally, comparing the fraction of the topology not discovered during inference testing by varying the network generation model is another topic for future work. Do certain network models better lend themselves to being discovered by topology inference tools? Conversely, are some network generation models less amenable to discovery?

Source of Loss Determination

In Section 5.1.1, we discuss an ASN in an emulated topology that experiences a significant amount of routing table churn, causing it to miss a substantial portion of the network during the link failures scenario. We believe it useful to determine whether the root cause of these missed ASs during inference probing is due to an incorrect FIB, or whether a FIB entry did not exist for the probe's destination prefix.

Emulated Topology Convergence

Throughout our experiments, we allow a fixed period of time to expire before beginning to probe our emulated topology in the first and third rounds. This is done to allow the routing tables of the routers in the network to converge; we rely, however, on our experience in determining the requisite amount of time this requires and add several more minutes for increased certainty. Because the vast majority of the time elapsed during our exhaustive network inference testing is spent during these wait periods, developing an accurate mechanism for determining whether routing table convergence has been achieved would undoubtedly improve efficiency.

Inference Under Different Fault Scenarios

In this thesis, we concern ourselves only with link failures that occur after a set interval has elapsed in our second probing round. One area for future research could involve spreading these failures over an interval of time according to some probability distribution, with random seeding for determinism and reproducibility. Alternatively, rather than use edge betweenness centrality as the metric for link failure selection, another measure could be utilized. Another possibility is to study router failures in which a router ceases to operate entirely or reboots, or perhaps just a number of interfaces on a particular router stop functioning. These faults could also be combined, in order to simulate a scenario in which failures are not constrained to one particular type.

Integration into Topology Deception Work

Much work has been done in the field of network topology deception – causing incoming probes to respond to their source with misleading responses, which in turn influences the inferred path as viewed from the source. Techniques for implementing this deception center around returning replies to probes from a single host, rewriting certain fields of the IP header according to a predetermined set of rules to force a particular inferred path on the probe source. With ERIK, however, traffic could be forwarded into the emulated network and achieve a similar purpose, eliminating the need to return probes from a host. Further, the emulated network used for this deception could be reconfigured at certain intervals, which would increase the difficulty for the probe source to determine that deception is occurring when compared to a static, standard reply.

Internet Graph Sampling Improvements

The graph sampling algorithms described in [27] rely upon random selections of nodes and edges to achieve a reduction of graph order. We introduce two new and novel methods for reducing AS-level Internet graphs in Chapter 4 that leverage graph properties – namely, the k-core of the graph. We believe that further study into utilizing Internet graph properties provides the most potential for sampling the Internet, and that new reduction algorithms should seek to exploit these graph metrics.

Internet Graph Prediction

We believe that experimenting with the inverse process of our reduction algorithms, or graph construction, may prove fruitful. That is, given the set of graph sampling methods studied in Chapter 2 and Chapter 4, applying the opposite logic and increasing the number of nodes and edges in a initial, chronologically earlier Internet instance. This methodology may provide a means to effectively create larger Internet approximations, and therefore be useful for predicting the growth of the Internet at the AS-level.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 106–114, 1988.

[2] RouteViews, Oregon, "University of Oregon RouteViews project," *Eugene, OR.[Online]. Available: http://www. routeviews. org*.

[3] D. G. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan, "Topology Inference from BGP Routing Dynamics," in *Proceedings of the 2nd SIGCOMM Workshop on Internet Measurement*.

[4] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet Map Discovery," in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.*, vol. 3. IEEE, 2000, pp. 1371–1380.

[5] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[6] A. Bender, R. Sherwood, and N. Spring, "Fixing Ally's Growing Pains with Velocity Modeling," in *Proceedings of the 8th SIGCOMM Conference on Internet Measurement*. ACM, 2008, pp. 337–342.

[7] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proceedings of the 6th SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 153–158.

[8] P. Mérindol, B. Donnet, J.-J. Pansiot, M. Luckie, and Y. Hyun, "MERLIN: MEasure the Router Level of the INternet," in *Proceedings of the 2011 7th EURO-NGI Conference on Next Generation Internet (NGI)*. IEEE, 2011, pp. 1–8.

[9] M. Luckie, "Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet," in *Proceedings of the 10th SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 239–245.

[10] P. Mérindol, B. Donnet, O. Bonaventure, and J.-J. Pansiot, "On the Impact of Layer-2 on Node Degree Distribution," in *Proceedings of the 10th SIGCOMM Conference on Internet Measurement*. ACM, 2010, pp. 179–191.

[11] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing Flash Crowds on the Internet," in *Modeling, Analysis and Simulation of Computer*

Telecommunications Systems, 2003. MASCOTS 2003. 11th International Symposium on. IEEE, 2003, pp. 246–249.

[12] G. F. Riley and T. R. Henderson, *The NS-3 Network Simulator*. New York City, NY: Springer, 2010.

[13] X. Chang, "Network Simulations with OPNET," in *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future-Volume 1*. ACM, 1999, pp. 307–314.

[14] M. Pizzonia and M. Rimondini, "Netkit: Easy Emulation of Complex Networks on Inexpensive Hardware," in *Proceedings of the 4th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 7.

[15] S. Knight, A. Jaboldinov, O. Maennel, I. Phillips, and M. Roughan, "AutoNetkit: Simplifying Large Scale, Open-Source Network Experimentation," in *Proceedings of the SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM, 2012, pp. 97–98.

[16] S. Knight, "Automated Configuration and Measurement of Emulated Networks with AutoNetkit," *SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 471–472, 2013.

[17] H. Nguyen, M. Roughan, S. Knight, N. Falkner, O. Maennel, and R. Bush, *How to Build Complex, Large-Scale Emulated Networks*. New York City, NY: Springer, 2011.

[18] J. Obstfeld, S. Knight, E. Kern, Q. S. Wang, T. Bryan, and D. Bourque, "VIRL: the Virtual Internet Routing Lab," in *Proceedings of the 2014 Conference on SIGCOMM*. ACM, 2014, pp. 577–578.

[19] S.-H. Yook, H. Jeong, and A.-L. Barabási, "Modeling the Internet's Large-Scale Topology," *Proceedings of the National Academy of Sciences*, vol. 99, no. 21, pp. 13 382–13 386, 2002.

[20] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems*. IEEE, 2001, pp. 346–353.

[21] C. Jin, Q. Chen, and S. Jamin, “Inet: Internet Topology Generator,” 2000, University of Michigan, Ann Arbor, MI. [Online]. Available: http://topology.eecs.umich.edu/inet/inet-2.0.pdf.

[22] J. P. Sterbenz, J. P. Rohrer, and E. K. Çetinkaya, “Multilayer Network Resilience Analysis and Experimentation on GENI,” The University of Kansas, Lawrence, KS, ITTC Technical Report ITTC-FY2011-TR-61349-01, July 2010. [Online]. Available: http://www.ittc.ku.edu/resilinets/reports/Path_Diversity_Experiments_TR_public.pdf

[23] M. Faloutsos, P. Faloutsos, and C. Faloutsos, “On Power-Law Relationships of the Internet Topology,” in *SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[24] Q. Chen, H. Chang, R. Govindan, and S. Jamin, “The Origin of Power Laws in Internet Topologies Revisited,” in *INFOCOM 2002. Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2. IEEE, 2002, pp. 608–617.

[25] L. Li, D. Alderson, W. Willinger, and J. Doyle, “A First-Principles approach to Understanding the Internet’s Router-Level Topology,” *SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 3–14, 2004.

[26] N. Spring, R. Mahajan, and D. Wetherall, “Measuring ISP Topologies with Rocketfuel,” in *SIGCOMM Computer Communication Review*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[27] V. Krishnamurthy, M. Faloutsos, M. Chrobak, J.-H. Cui, L. Lao, and A. G. Percus, “Sampling Large Internet Topologies for Simulation Purposes,” *Computer Networks*, vol. 51, no. 15, pp. 4284–4302, 2007.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein *et al.*, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2001, vol. 2.

[29] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring Network Structure, Dynamics, and Function using NetworkX,” in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA, Aug. 2008, pp. 11–15.

[30] Dynamips Github Issues Board, July 2014, Dynamips. [Online]. Available: https://github.com/GNS3/dynamips/issues/50.

[31] Dynamips Github Issues Board, Nov. 2014, Dynamips. [Online]. Available: https://github.com/GNS3/dynamips/issues/59.

[32] M. Newman, *Networks: an Introduction*. Oxford, United Kingdom: Oxford University Press, 2010.

[33] M. Bastian, S. Heymann, M. Jacomy *et al.*, “Gephi: an Open Source Software for Exploring and Manipulating Networks,” *ICWSM*, vol. 8, pp. 361–362, 2009.

[34] J. I. Alvarez-Hamelin, L. Dall’Asta, A. Barrat, and A. Vespignani, “k-core Decomposition of Internet Graphs: Hierarchies, Self-Similarity and Measurement Biases,” *arXiv preprint cs/0511007*, 2005.

[35] C. Gkantsidis, M. Mihail, and E. Zegura, “Spectral Analysis of Internet Topologies,” in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 1. IEEE, 2003, pp. 364–374.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California